University of Ljubljana
Faculty of Computer and Information Science

Algorithm analysis: Exercises with solutions for AAHPS tutorials

Matej Pičulin

Ljubljana, 2022

Contents

1 Introduction note

A selection of exercises and solutions for the course Algorithm Analysis and Heuristic Problem solving that were used over past few years. Some exercises are taken from the book Introduction to Algorithms, 3rd Ed., by Cormen, T. H., et al., Massachusetts London, England, 2009 and from Jeff Erickson's Appending available at `https://courses.engr.illinois.edu/cs573/fa2010/notes/99-recurrences.pdf`.

This script covers recurrences, probabilistic and amortized analysis.

2 Time complexity review

2.1 Time complexity of code snippets

Find the (exact) number of times *print()* function executes for the following loops using n = 10 and n = 20. Next, find or estimate tight asymptotic bound of the following loops. If you can't find tight asymptotic bound find lower and upper asymptotic bound. Assume that all undefined variables used are of type integer.

Algorithm 1

```
1: function FUN(int n)
2:     for (int i = 0; i < n; i++) do
3:         Console.print(i)
4:     end for
5:     return 0
6: end function
```

Solution 1: For loop executes print() statement exactly n times. So for n = 10 and 20 the answer is 10 and 20. Asymptotically tight bound is $\Theta(n)$.

Algorithm 2

```
1: function FUN(int n)
2:     for (int i = 1; i < n/2; i+=5) do
3:         Console.print(i)
4:     end for
5:     return n
6: end function
```

Solution 2: For loop moves by 5 each step and only goes to half n, meaning print() function executes $\frac{n}{10}$ times. So for $n = 10$ and 20 the answer is 1 and 2. Asymptotically tight bound is still $\Theta(n)$.

Algorithm 3

```
1: function FUN(int n)
2:     for (int i = 0; i < n; i++) do
3:         Console.print(i)
4:     end for
5:     for (int i = n; i > 0; i– –) do
6:         Console.print(i)
7:     end for
8:     return 0
9: end function
```

Solution 3: Each for loop executes print() statement exactly n times. So for $n = 10$ and 20 the answer is 20 and 40. Asymptotically tight bound is also $\Theta(n)$.

Algorithm 4

```
1: function FUN(int n)
2:     for (int i = n; i > 1; i/=2) do
3:         Console.print(i)
4:     end for
5:     return 0
6: end function
```

Solution 4: For loop halves the value of i in each iteration, meaning the print() statement executes $\log(n)$ times. So for $n = 10$ and 20 the answer is 3 and 4. Asymptotically tight bound is $\Theta(\log(n))$.

Algorithm 5

```
1: function FUN(int n)
2:     for (int i = 0; i < n/2; i++) do
3:         for (int j = 0; j < n/2; j++) do
4:             Console.print(i+j)
5:         end for
6:     end for
7:     return 0
8: end function
```

Solution 5: The outer loop executes $\frac{n}{2}$ times and for each repetition of outer loop the inner one also executes $\frac{n}{2}$ times. Meaning the print() statement executes $\frac{n^2}{4}$ times. So for $n = 10$ and 20 the answer is 25 and 100. Asymptotically tight bound is therefore $\Theta(n^2)$.

Algorithm 6

```
1: function FUN(int n)
2:     int m = 2000
3:     for (int i = 0; i < n/2; i++) do
4:         for (int j = 0; j < m*1000; j++) do
5:             Console.print(i+j)
6:         end for
7:     end for
8:     return 0
9: end function
```

Solution 6: Again we have a nested loop but in this case we can assume m to be constant. Meaning the print() statement executes $\frac{n}{2} \cdot 2000 \cdot 1000$ times. So for $n = 10$ and 20 the answer is 10 million and 20 million times. Asymptotically tight bound is $\Theta(n)$.

Algorithm 7

```
1: function FUN(int n)
2:     int m = n/10000
3:     for (int i = 0; i < n/2; i++) do
4:         for (int j = 0; j < m; j+=20) do
5:             Console.print(i+j)
6:         end for
7:     end for
8:     return 0
9: end function
```

Solution 7: Similar to the previous one, except the variable m is now not a constant since it is depended on n. Meaning the print() statement executes $\frac{n}{2} \cdot \frac{n}{20 \cdot 10000}$ times. So for $n = 10$ and 20 the answer is both times 0. Asymptotically tight bound is $\Theta(n^2)$, even though practically this algorithm runs a lot faster than Algorithm 6.

Algorithm 8

```
1: function FUN(int n)
2:     for (int i = 0; i < n; i++) do
3:         for (int j = i; j < n; j++) do
4:             Console.print(n)
5:         end for
6:     end for
7:     return 0
8: end function
```

Solution 8: This nested loop is a bit harder to analyse since the number of repetitions of inner loop depends on the value of i of outer loop. We can see that for $i = 0$ the inner loop executes n times, for $i = 1$ $n - 1$ times etc. All together $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$ times. So for $n = 10$ and 20 the answer is 55 and 210. Asymptotically tight bound is $\Theta(n^2)$.

Algorithm 9

```
1: function FUN(int n)
2:     int m = n*n;
3:     for (int i = n/2; i > 1; i/=3) do
4:         for (int j = 0; j < m; j++) do
5:             Console.print(i+j)
6:         end for
7:     end for
8:     return 0
9: end function
```

Solution 9: This is a combination of previous algorithms. Variable m is not a constant and is quadratically depended on n. The outer loop executes $\log_3(\frac{n}{2})$ times and the inner loop n^2 times. All together $n^2 \log_3(\frac{n}{2})$ times. So for $n = 10$ and 20 the answer is 100 and 800. Asymptotically tight bound is $\Theta(n^2 \log(n))$.

Notice that the base of logarithm isn't important for apocalyptical notation since the translation between different bases differs only for a constant.

Algorithm 10

```
1: function FUN(int n)
2:     int m = 2147483647; //MAX_INTEGER
3:     for (int i = 0; i < m; i++) do
4:         for (int j = m; j < m; j/=2) do
5:             Console.print(n)
6:         end for
7:     end for
8:     return 0
9: end function
```

Solution 10: In this case m is constant and the program is quite slow since the outer loop repeats many times. The print() statement never executes since in inner loop $j = m$ and is never lower. So for $n = 10$ and 20 the answer is both time 0. Asymptotically tight bound is $\Theta(1)$ since the running time of the algorithm is not dependent on n.

Algorithm 11

```
1: function FUN(int n)
2:     while n > 1 do
3:         Console.print(n)
4:         n = n / 2
5:     end while
6:     return 0
7: end function
```

Solution 11: We can analyse while loop the same as for loops. Here the variable n halves every iteration meaning asymptotically tight bound is $\Theta(\log(n))$. And for $n = 10$ and 20 the answer is 3 and 4.

Algorithm 12

```
1: function FUN(int n)
2:     m = 0
3:     while m * m < n + 100 do
4:         Console.print(m)
5:         m++
6:     end while
7:     return 0
8: end function
```

Solution 12: Here we can view m as a counter of while loop repetitions. Observe the stopping condition of the for loop $m^2 < n+100$. We need to extract m from equation and get $m = \sqrt{n + 100}$ which is also the number of while loop repetitions. So for $n = 10$ and 20 the answer both time 11. Asymptotically tight bound is $\Theta(\sqrt{n})$.

Algorithm 13

```
1: function FUN(int n)
2:     while n > 0 do
3:         for (int i = n; i > 0; i/=3) do
4:             Console.print(n)
5:         end for
6:         n = n / 2
7:     end while
8:     return 0
9: end function
```

Solution 13: The outer loop repeats $\log(n)$ times. The inner loop repeats $\log_3(n)$ times. So for $n = 10$ and 20 the answer is 7 and 10. Asymptotically tight bound is $\Theta((\log(n))^2) = \Theta(\log^2(n))$. **Note:** Inner loop is actually depended on outer loop since n changes. But even with summing the exact values we would still have the same asymptotic tight bound.

Algorithm 14

```
1: function FUN(int n)
2:     while n > 0 do
3:         for (int i = n; i > 1; i = sqrt(i)) do
4:             Console.print(n)
5:         end for
6:         n = n / 2
7:     end while
8:     return 0
9: end function
```

Solution 14: As in previous example we will ignore that the inner loop is depended on outer. The outer loop repeats $\log(n)$ times. The number of repetitions for inner loop is harder to estimate. The variable i drops very fast. Lets take for example that the n is initially 65536. The sequence for i is therefore 65536, 256, 16, 4, 2, 1. Lets rewrite this sequence:

$$\begin{aligned}\sqrt{65536} = 256 = 2^8 = 2^{2^3}\\ \sqrt{256} = 16 = 2^4 = 2^{2^2}\\ \sqrt{16} = 4 = 2^2 = 2^{2^1}\\ \sqrt{4} = 2 = 2^1 = 2^{2^0}\\ \sqrt{2} = 1 = 2^0\end{aligned}$$

You can notice that the sequence is 2^{2^i} and inverse of this is $\log(\log(n))$. This means that asymptomatic tight bound is $\Theta(\log(n) \cdot \log(\log(n)))$. So for n = 10 and 20 the answer is 5 and 8.

Algorithm 15

```
1: function FUN(int n)
2:     m = n * n
3:     while m > 2 do
4:         Console.print(n)
5:         m = m / 2
6:     end while
7:     return 0
8: end function
```

Solution 15: This is quite easy exercise. Variable m equals n^2. Meaning the print() statement repeats $\log(n^2)$ times. So for $n = 10$ and 20 the answer is 6 and 8. Asymptotically tight bound is $\Theta(\log(n^2)) = \Theta(log(n))$. **Note:** $\log(n^a) = a\log(n)$

Algorithm 16

```
1: function FUN(int n, int m)
2:     while n > 0 do
3:         for (int i = 0; i < m; i++) do
4:             Console.print(n)
5:         end for
6:         n = n / 2
7:     end while
8:     return 0
9: end function
```

Solution 16: In this case m is also input parameter and we usually estimate the running time of an algorithm based on all the inputs. Meaning the print() statement repeats $m\log(n)$ times. So for $n = m = 10$ and 20 the answer is 40 and 100. Asymptotically tight bound is $\Theta(m\log(n))$.

Algorithm 17 Random() returns a number between 0 and 1

```
1: function FUN(int n)
2:     int m = 0
3:     for (int i = 0; i < n; i++) do
4:         m += i
5:     end for
6:     if Random() > 0.5 then
7:         return 0
8:     end if
9:     for (int i = 0; i < m * n; i++) do
10:        Console.print("?")
11:    end for
12:    return 0
13: end function
```

Solution 17: In this example the exact number of print() statements is dependent on a random function Random(). The print() statement might not run and in this case the asymptotic bound is $\Omega(1)$. Assuming Random() returns ≤ 0.5 the print() executes $m*n$ times where $m = \frac{n(n+1)}{2}$). So for $n = 10$ and 20 the answer is either 0 if only first for loop execute or 450 and 3800, respectively if both loops execute. From this we don't know what asmptiotic tight bound actually is but we can estimate lower as $\Omega(1)$ and upper bound as $O(n^3)$.

Algorithm 18

```
1: function FUN(int n, int m)
2:     if n > m then
3:         return 0
4:     end if
5:     for (int i = 0; i < n; i++) do
6:         for (int j = i; j < m; j++) do
7:             Console.print(" :)")
8:         end for
9:     end for
10:    return 0
11: end function
```

Solution 18: In this case m is again an input parameter. The number of print() statements is in this case depends on inputs and varies between 0 and $\frac{m(m1+)}{2}$. We can analyse this in few ways. If we ignore the dependence of inner loop on the outer we can say that the asymptotic running time is $O(nm)$. If the print() statement executes we know that $n > m$ leading to $O(n^2)$. But since we are estimating the number of times print() statements executes we can see that the print() only executes when $i < m$ otherwise the inner loop is skipped. This leads to $O(m^2)$. So for $n = m = 10$ and 20 the answer is 0. Lower bound is again $\Omega(1)$.

2.2 Streaming median

Below are two streaming median algorithms. The streaming median algorithm keeps track of the median even if we add numbers to the list. Find the upper asymptotic bound of both algorithms.

Structure List consists of two functions. Function **add**(int n) adds a new element n in sorted order to the structure in $\Theta(n)$. Function **get**(int i) returns i-th element in constant time.

Structure Heap has four functions. **Insert()** inserts new element in structure in $\Theta(log(n))$. Function **size()** returns the number of elements in heap in constant time. Function **peek()** returns minimal (or maximal) element in the structure in constant time. Functions **extractMin()** and **extractMax()** return and delete minimal/maximal element respectively in $\Theta(log(n))$ time.

Algorithm 19

```
Require: external List sortedNumbers
1: function findMedian(int newNumber)
2:     sortedNumbers.add(newNumber)
3:     int medianIndex = length(sortedNumbers)/2
4:     return sortedNumbers.get(medianIndex)
5: end function
```

Solution 19: The solution is straightforward. Line 2 takes $\Theta(n)$ time, line 3 $\Theta(1)$ and line 4 $\Theta(1)$. All together is $\Theta(n)$.

Algorithm 20

```
Require: external Heap minHeap, maxHeap
1: function findMedian(int newNumber)
2:     if newNumber < maxHeap.peek() then
3:         maxHeap.insert(newNumber)
4:     else
5:         minHeap.insert(newNumber)
6:     end if
7:     if minHeap.size()+1 > maxHeap.size() then
8:         int root = minHeap.extractMin()
9:         maxHeap.insert(root)
10:    else if maxHeap.size()+1 > minHeap.size() then
11:        int root = maxHeap.extractMax()
12:        maxHeap.insert(root)
13:    end if
14:    if minHeap.size() >= maxHeap.size() then
15:        return minHeap.peek()
16:    else
17:        return maxHeap.peek()
18:    end if
19: end function
```

Solution 20: First lets look at lines 2-6. The condition of if statement takes $\Theta(1)$ time and both lines 3 or 5 $\Theta(\log(n))$. Lines 7-13 are similar. The conditions take $\Theta(1)$ time and both bodies of if statements $\Theta(\log(n) + \log(n))$. Together

these lines take $O(\log(n))$ or $\Omega(1)$ if *maxHeap.size() == minHeap.size()*. Lines 14-18 all take $\Theta(1)$. We can see that the most time consuming part are lines 2-6 leading asymptotically leading to $\Theta(\log(n))$.

2.3 Sorting functions

Sort the following functions in ranks. From lowest rank to highest rank depending on asymptotic growth.

- n
- $nlog(n)$
- $2^{log_2(n)}$
- 2^n
- 3^n
- 2^{n+7}
- $log(n^2)$
- $3\sqrt{n}$
- $n!$
- n^2
- $\binom{n}{2}$
- 2^{4096}
- n^3
- $log(log(n))$
- $(n-1)!$
- $7n^3+5n^2-2n+13$
- $n \cdot 2^n$
- $log\big(\sqrt{log(n)}\big)$
- $n \cdot log(n^n)$

Solution: From asymptotically slowest function on top to fastest on bottom. Some functions only differ for a constant.

- $n!$
- $(n-1)!$
- 3^n
- $n \cdot 2^n$
- 2^n and 2^{n+7}
- n^3 and $7n^3+5n^2-2n+13$
- $n \cdot log(n^n)$
- n^2 and $\binom{n}{2}$
- $nlog(n)$
- n and $2^{log_2(n)}$
- $3\sqrt{n}$
- $log(n^2)$
- $log(log(n))$ and $log\big(\sqrt{log(n)}\big)$
- 2^{4096}

If you are unsure which function is slower you can test using $\lim_{n\to\text{inf}} \frac{f(n)}{g(n)}$. If the result is 0 then $\Theta(g(n)) > \Theta(f(n))$, if result is a constant c then $\Theta(g(n)) = \Theta(f(n))$ and if the result is inf then $\Theta(g(n)) < \Theta(f(n))$.

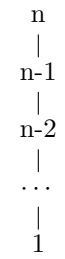
3 Solving recurrences

3.1 Tree method

Approximate upper and lower asymptotic bound of the following recurrences.

$$T(n) = T(n-1) + n \tag{1}$$

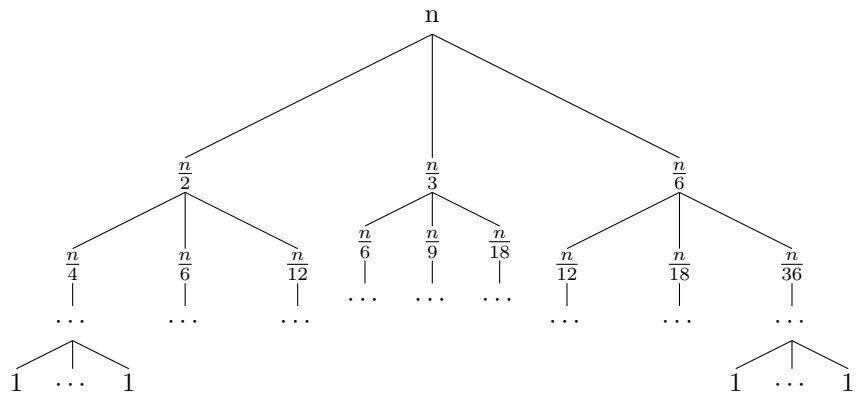
Solution:



Sums by levels: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{3}) + T(\frac{n}{6}) + n \tag{2}$$

Solution:



Sums by level:

$$
\begin{aligned}
&Level : Sum \\
&\quad 1 : n \\
&\quad 2 : \frac{n}{2} + \frac{n}{3} + \frac{n}{6} = \\
&\quad\quad = \frac{3n}{6} + \frac{2n}{6} + \frac{n}{6} = n \\
&\quad 3 : \frac{n}{4} + \frac{n}{6} + \frac{n}{12} + \\
&\quad\quad + \frac{n}{6} + \frac{n}{9} + \frac{n}{18} + \\
&\quad\quad + \frac{n}{12} + \frac{n}{18} + \frac{n}{36} = \\
&\quad\quad = \frac{9n + 6n + 3n + 6n + 4n + 2n + 3n + 2n + n}{36} = \\
&\quad\quad = n \\
&\quad 4 : \cdots
\end{aligned}
$$

Max depth of tree is $log_6(n)$, meaning we can estimate upper asymptotic bound with $O(n \cdot log(n))$.

Leaves have a cost of T(1). Assuming the max depth of $log_6(n)$ we have $3^{log_6(n)} = n^{log_6(3)} = n^{0.613}$ leaves giving us upper estimate of $O(n^{log_6(3)})$ which is lower than $O(n \cdot log(n))$.

$$T(n) = T(\frac{n}{2}) + n^2 \tag{3}$$

Solution:

$$
\begin{array}{c}
\mathrm{n}^2 \\
| \\
\frac{n^2}{2} \\
| \\
\frac{n^2}{4} \\
| \\
\cdots \\
| \\
1
\end{array}
$$

Sums by level:

$$
\begin{aligned}
&Level : Sum \\
&\quad 1 : n^2 \\
&\quad 2 : \frac{n^2}{2} \\
&\quad 3 : \frac{n^2}{4} \\
&\quad \vdots \\
&\quad i : \frac{n^2}{2^{i-1}}
\end{aligned}
$$

3.2 Master method

Using the master method solve the following recurrences.

$$T(n) = 2 \cdot T(\frac{n}{4}) + 1 \tag{4}$$

Solution : Using simplified Masters method:

$$a = 2, b = 4, d = 0$$
$$\text{Case 1: } a > b^d \rightarrow 2 > 4^0$$
$$\Theta(n^{log_b a}) = \Theta(\sqrt{(n)})$$

$$T(n) = 2 \cdot T(\frac{n}{4}) + \sqrt{n} \tag{5}$$

Solution : Using simplified Masters method:

$$a = 2, b = 4, d = \frac{1}{2}$$
$$\text{Case 2: } a = b^d \rightarrow 2 > 4^{\frac{1}{2}}$$
$$\Theta(n^d log_b n) = \Theta(\sqrt{(n)} log(n))$$

$$T(n) = 2 \cdot T(\frac{n}{4}) + n \tag{6}$$

Solution : Using simplified Masters method:

$$a = 2, b = 4, d = 1$$
$$\text{Case 3: } a < b^d \rightarrow 2 < 4^1$$
$$\Theta(n^d) = \Theta(n)$$

$$T(n) = 2 \cdot T(\frac{n}{4}) + n^2 \tag{7}$$

Solution : Using simplified Masters method:

$$a = 2, b = 4, d = 2$$
$$\text{Case 3: } a < b^d \rightarrow 2 < 4^2$$
$$\Theta(n^d) = \Theta(n^2)$$

$$T(n) = 7 \cdot T(\frac{n}{2}) + n^2 \tag{8}$$

Solution : Using simplified Masters method:

$$a = 7, b = 2, d = 2$$
$$\text{Case 1: } a > b^d \rightarrow 7 > 2^2$$
$$\Theta(n^{log_b a}) = \Theta(n^{log_2 7}) \approx \Theta(n^{2.81})$$

$$T(n) = 4 \cdot T(\frac{n}{2}) + n^2 lg(n) \tag{9}$$

Solution : Using Masters method:

$$a = 4, b = 2, f(n) = n^2 \log(n)$$
$$n^{\log_b a} = n^2$$
$$\text{Case 2 extended: } f(n) = \Theta(n^2 \log^1(n)) \rightarrow T(n) = \Theta(n^2 \log^2 n)$$

3.3 Akra-Bazzi

Find tight asymptotic bounds of the following recursive functions.

$$T(n) = 2 \cdot T\big(\frac{n}{2}\big) + n \tag{10}$$

Solution : Using Akra-Bazzi:

$$a_1 = 2, b_1 = \frac{1}{2}, f(n) = n$$
$$a_1 \cdot b_1^p = 1 \rightarrow p = 1$$
$$T(n) = \Theta(n(1 + \int_1^n \frac{u}{u^2} du))$$
$$= \Theta(n(1 + \int_1^n \frac{1}{u} du))$$
$$= \Theta(n(1 + \log u\big|_1^n))$$
$$= \Theta(n(1 + \log(n) - \log(1)))$$
$$= \Theta(n \log(n))$$

$$T(n) = 2 \cdot T\big(\frac{n}{2}\big) + n \cdot lg(n) \tag{11}$$

Solution : Using Akra-Bazzi:

$$
\begin{aligned}
a_1 = 2, b_1 = \frac{1}{2}, f(n) &= n \cdot \log(n) \\
a_1 \cdot b_1^p = 1 &\to p = 1 \\
T(n) = \Theta(n(1 + \int_1^n \frac{u \log(u)}{u^2} du)) & \\
= \Theta(n(1 + \int_1^n \frac{\log(u)}{u} du)) & * \\
= \Theta(n(1 + \frac{\log^2(u)}{2})) & \\
= \Theta(n \log^2(n)) &
\end{aligned}
$$

$$
\begin{aligned}
x = \log(u) \to dx = \frac{1}{u} du & \\
dx = \frac{du}{u} \to x = log(n) & \\
\int_1^n \frac{\log(u)}{u} du)) = & \\
= \int_0^{\log(u)} x dx = \frac{x^2}{2}\Big|_0^{\log(u)} = & \\
= \frac{\log^2(u)}{2} &
\end{aligned}
$$

$$
T(n) = T\big(\Big\lceil \frac{n}{2} \Big\rceil\big) + T\big(\Big\lfloor \frac{n}{2} \Big\rfloor\big) + n \tag{12}
$$

Solution : Using extended Akra-Bazzi:

First rewrite the equation to:

$$
T(n) = T\big(\frac{n}{2} + \Big\lceil \frac{n}{2} \Big\rceil - \frac{n}{2}\big) + T\big(\frac{n}{2} + \Big\lfloor \frac{n}{2} \Big\rfloor - \frac{n}{2}\big) + n
$$

$$
\begin{aligned}
a_1 = 1, a_2 = 1, b_1 = \frac{1}{2}, b_2 = \frac{1}{2}, f(n) = n \\
h_1(n) = \left\lceil \frac{n}{2} \right\rceil - \frac{n}{2}, h_2(n) \left\lfloor \frac{n}{2} \right\rfloor - \frac{n}{2} \\
h_1(n) \in [0,1], h_2(n) \in [-1,0] \\
a_1 \cdot b_1^p + a_2 \cdot b_2^p = 1 \rightarrow p = 1 \\
T(n) = \Theta(n(1 + \int_1^n \frac{u}{u^2} du)) \\
= \Theta(n(1 + \int_1^n \frac{1}{u} du)) \\
= \Theta(n(1 + \log u \Big|_1^n)) \\
= \Theta(n(1 + \log(n) - \log(1))) \\
= \Theta(n \log(n))
\end{aligned}
$$

$$
T(n) = 2 \cdot T(\frac{n}{2}) + \frac{9}{2} \cdot T(\frac{n}{3}) + \theta(n) \tag{13}
$$

Solution : Using Akra-Bazzi:

$$
\begin{aligned}
a_1 = 2, a_2 = \frac{9}{2}, b_1 = \frac{1}{2}, b_2 = \frac{1}{3}, f(n) = \Theta(n) \\
a_1 \cdot b_1^p + a_2 \cdot b_2^p = 1 \rightarrow p = 2 \\
T(n) = \Theta(n^2(1 + \int_1^n \frac{u}{u^3} du)) \\
= \Theta(n^2(1 + \int_1^n \frac{1}{u^2} du)) \\
= \Theta(n^2(1 + \frac{u^{-1}}{-1} \Big|_1^n)) \\
= \Theta(n^2(1 - \frac{1}{n} + 1)) \\
= \Theta(n^2)
\end{aligned}
$$

$$
T(n) = 2 \cdot T(\frac{n}{4}) + 3 \cdot T(\frac{n}{6}) + n \cdot lg(n) \tag{14}
$$

Solution : Using Akra-Bazzi:

$$a_1 = 2, a_2 = 3, b_1 = \frac{1}{4}, b_2 = \frac{1}{6}, f(n) = n\log(n)$$
$$a_1 \cdot b_1^p + a_2 \cdot b_2^p = 1 \rightarrow p = 1$$
$$T(n) = \Theta(n(1 + \int_1^n \frac{u\log(u)}{u^2} du))$$
$$= \Theta(n(1 + \int_1^n \frac{\log(u)}{u}* du))$$
$$= \Theta(n(1 + \frac{\log^2(u)}{2}))$$
$$= \Theta(n\log^2(n))$$

* See solution to equation 13.

3.4 Annihilators

For each of the following recurrences find the closed form solution, estimate upper asymptotic bound and prove your solution is correct.

$$T(n) = T(n-1) + 1; T(0) = 0 \tag{15}$$

Solution : Using Annihilators:

$$\text{Step 1: } T(n) = T(n-1) + 1; T(0) = 0$$
$$T(n+1) = T(n) + 1$$
$$T(n+1) - T(n) = 1$$
$$ET(n) - T(n) = 1$$
$$(E-1)T(n) = 1$$
$$\text{Step 2: } (E-1) \Leftrightarrow (E-1) \rightarrow (E-1)^2$$
$$\text{Step 3: Already factored.}$$
$$\text{Step 4: } T(n) = \alpha n + \beta$$
$$\text{Step 5: } T(0) = \alpha n + \beta = 0 = \beta$$
$$T(1) = \alpha n + \beta = 1 = \alpha$$
$$T(n) = n$$

$$T(n) = T(n-1) + n; T(1) = 1 \tag{16}$$

Solution : Using Annihilators:

$$\begin{aligned}
\text{Step 1: } T(n) &= T(n-1) + n; T(1) = 1 \\
T(n+1) &= T(n) + n + 1 \\
T(n+1) - T(n) &= n + 1 \\
ET(n) - T(n) &= n + 1 \\
(E-1)T(n) &= n + 1 \\
\text{Step 2: } (E-1) &\Leftrightarrow (E-1)^2 \rightarrow (E-1)^3 \\
\text{Step 3: Already factored.} & \\
\text{Step 4: } T(n) &= \alpha_2 n^2 + \alpha_1 n + \alpha_0 \\
\text{Step 5: } T(1) &= 1 = \alpha_2 + \alpha_1 + \alpha_0 \\
T(2) &= 3 = 4\alpha_2 + 2\alpha_1 + \alpha_0 \\
T(3) &= 6 = 9\alpha_2 + 3\alpha_1 + \alpha_0 \\
\text{Solution : } \alpha_2 &= \frac{1}{2}, \alpha_1 = \frac{1}{2}, \alpha_0 = 0 \\
T(n) &= \frac{1}{2}n^2 + \frac{1}{2}n
\end{aligned}$$

$$T(n) = T(n-1) + 2 \cdot n + 1; T(0) = 0 \tag{17}$$

Solution : Using Annihilators:

$$\begin{aligned}
\text{Step 1: } T(n) &= T(n-1) + 2n + 1; T(0) = 0 \\
T(n+1) &= T(n) + 2n + 3 \\
T(n+1) - T(n) &= 2n + 3 \\
ET(n) - T(n) &= 2n + 3 \\
(E-1)T(n) &= 2n + 3 \\
\text{Step 2: } (E-1) &\Leftrightarrow (E-1)^2 \rightarrow (E-1)^3 \\
\text{Step 3: Already factored.} & \\
\text{Step 4: } T(n) &= \alpha_2 n^2 + \alpha_1 n + \alpha_0 \\
\text{Step 5: } T(0) &= 0 = \alpha_0 \\
T(1) &= 3 = \alpha_2 + \alpha_1 \\
T(2) &= 8 = 4\alpha_2 + 2\alpha_1 \\
\text{Solution : } \alpha_2 &= 2, \alpha_1 = 1, \alpha_0 = 0 \\
T(n) &= 2n^2 + n
\end{aligned}$$

$$T(n) = T(n-1) + \binom{n}{2}; T(0) = 0 \tag{18}$$

Solution : Using Annihilators:

$$\begin{aligned}
\text{Step 1: } T(n) &= T(n-1) + \binom{n}{2}; T(0) = 0 \\
T(n+1) &= T(n) + \binom{n+1}{2} \\
T(n+1) - T(n) &= \binom{n+1}{2} \\
ET(n) - T(n) &= \frac{n^2+n}{2} \\
(E-1)T(n) &= \frac{n^2+n}{2} \\
\text{Step 2: } (E-1) &\Leftrightarrow (E-1)^3 \rightarrow (E-1)^4 \\
\text{Step 3: Already factored.} & \\
\text{Step 4: } T(n) &= \alpha_3 n^3 + \alpha_2 n^2 + \alpha_1 n + \alpha_0 \\
\text{Step 5: } T(0) &= 0 = \alpha_0 \\
T(1) &= 0 = \alpha_3 + \alpha_2 + 2\alpha_1 \\
T(2) &= 1 = 8\alpha_3 + 4\alpha_2 + 2\alpha_1 \\
T(3) &= 4 = 27\alpha_3 + 9\alpha_2 + 3\alpha_1 \\
\text{Solution : } \alpha_3 &= \frac{1}{6}, \alpha_2 = 0, \alpha_1 = -\frac{1}{6}, \alpha_0 = 0 \\
T(n) &= \frac{1}{6}n^3 - \frac{1}{6}n
\end{aligned}$$

$$T(n) = T(n-1) + 2^n; T(0) = 0 \tag{19}$$

Solution : Using Annihilators:

$$\begin{aligned}
\text{Step 1: } T(n) &= T(n-1) + 2^n; T(0) = 0 \\
T(n+1) &= T(n) + 2 \cdot 2^n \\
T(n+1) - T(n) &= 2 \cdot 2^n \\
ET(n) - T(n) &= 2 \cdot 2^n \\
(E-1)T(n) &= 2 \cdot 2^n \\
\text{Step 2: } (E-1) &\Leftrightarrow (E-2) \rightarrow (E-1)(E-2) \\
\text{Step 3: Already factored.} & \\
\text{Step 4: } T(n) &= \alpha + \beta 2^n \\
\text{Step 5: } T(0) &= 0 = \alpha + \beta \\
T(1) &= 2 = \alpha + 2\beta \\
\text{Solution : } \alpha &= -2, \beta = 2 \\
T(n) &= 2 \cdot 2^n - 2 = 2^{n+1} - 2
\end{aligned}$$

$$T(n) = 3 \cdot T(n-1); T(0) = 1 \tag{20}$$

Solution : Using Annihilators:

$$\begin{aligned}
\text{Step 1: } T(n) &= 3T(n-1); T(0) = 1 \\
T(n+1) &= 3T(n) \\
T(n+1) - 3T(n) &= 0 \\
ET(n) - 3T(n) &= 0 \\
(E-3)T(n) &= 0 \\
\text{Step 2: } (E-3) &\Leftrightarrow 1 \rightarrow (E-3) \\
\text{Step 3: Already factored.} & \\
\text{Step 4: } T(n) &= \alpha 3^n \\
\text{Step 5: } T(0) &= 1 = \alpha \\
\text{Solution : } \alpha &= \frac{1}{3} \\
T(n) &= \frac{1}{3} 3^n = 3^{n-1}
\end{aligned}$$

$$T(n) = 2 \cdot T(n-1) + 1; T(0) = 0 \tag{21}$$

Solution : Using Annihilators:

$$\begin{aligned}
\text{Step 1: } T(n) &= 2T(n-1) + 1; T(0) = 0 \\
T(n+1) &= 2T(n) + 1 \\
T(n+1) - 2T(n) &= 1 \\
ET(n) - 2T(n) &= 1 \\
(E-2)T(n) &= 1 \\
\text{Step 2: } (E-2) &\Leftrightarrow (E-1) \rightarrow (E-2)(E-1) \\
\text{Step 3: Already factored.} & \\
\text{Step 4: } T(n) &= \alpha + \beta 2^n \\
\text{Step 5: } T(0) &= 0 = \alpha + \beta \\
T(1) &= 1 = \alpha + 2\beta \\
\text{Solution : } \alpha &= -1, \beta = 1 \\
T(n) &= 2^n - 1
\end{aligned}$$

$$T(n) = T(n-1) + T(n-2) + 1; T(-1) = 0, T(0) = 1 \tag{22}$$

Solution : Using Annihilators:

Lets define golden ratio and its conjugate:

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618, \hat{\varphi} = \frac{1-\sqrt{5}}{2} \approx -0.618$$

$$\begin{aligned}
\text{Step 1: } T(n) &= T(n-1) + T(n-2) + 1; T(-1) = 0, T(0) = 1 \\
T(n+2) &= T(n+1) + T(n) + 1 \\
T(n+2) - T(n+1) - T(n) &= 1 \\
E^2T(n) - ET(n) - T(n) &= 1 \\
(E^2 - E - 1)T(n) &= 1 \\
\text{Step 2: } (E^2 - E - 1) &\Leftrightarrow (E-1) \rightarrow (E^2 - E - 1)(E-1) \\
\text{Step 3: } (E-\varphi)(E-\hat{\varphi})&(E-1) \\
\text{Step 4: } T(n) &= \alpha\varphi^n + \beta\hat{\varphi}^n + \gamma \\
\text{Step 5: } T(-1) &= 0 = \alpha\varphi^{-1} + \beta\hat{\varphi}^{-1} + \gamma \\
T(0) &= 1 = \alpha + \beta + \gamma \\
T(1) &= 2 = \alpha\varphi + \beta\hat{\varphi} + \gamma \\
\text{Solution : } \alpha &= \frac{5+2\sqrt{5}}{5}, \beta = \frac{5-2\sqrt{5}}{5}, \gamma = -1 \\
T(n) &= \frac{5+2\sqrt{5}}{5}\varphi^n + \frac{5-2\sqrt{5}}{5}\hat{\varphi}^n - 1
\end{aligned}$$

3.5 Substitution method

Guess and prove the upper asymptotic bounds of the following recurrences.

$$T(n) = T(\left\lfloor \frac{n}{2} \right\rfloor) + 1 \tag{23}$$

1.) Guess: $T(n) \leq c \cdot log_2(n)$
2.) Induction: $T(\lfloor \frac{n}{2} \rfloor) \leq c \cdot log_2 \lfloor \frac{n}{2} \rfloor$
3.) Proof:

$$\begin{aligned} T(n) &\leq c \cdot log_2(\lfloor \frac{n}{2} \rfloor) + 1 \\ &\leq c \cdot log_2(\frac{n}{2}) + 1 \\ &= c \cdot log_2(n) - c \cdot log_2(2) + 1 \\ &= c \cdot log_2(n) - c + 1; c \geq 1 \\ &\leq c \cdot log_2(n) \end{aligned}$$

$$T(n) = T(\left\lceil \frac{n}{2} \right\rceil) + 1 \tag{24}$$

1.) Guess: $T(n) \leq c \cdot log_2(n)$
2.) Induction: $T(\lceil \frac{n}{2} \rceil) \leq c \cdot log_2 \lceil \frac{n}{2} \rceil$
3.) Proof:

$$\begin{aligned} T(n) &\leq c \cdot log_2(\lceil \frac{n}{2} \rceil) + 1 \\ &\leq c \cdot log_2(\frac{n}{2} + 1) + 1 \\ &= c \cdot log_2(\frac{n+2}{2}) + 1 \\ &= c \cdot log_2(n+2) - c + 1; c > 1 \\ &\leq c \cdot log_2(n+2); deadend \end{aligned}$$

4.) New guess: $T(n) \leq c \cdot log_2(n - d)$
5.) Induction: $T(\lceil \frac{n}{2} \rceil) \leq c \cdot log_2 \lceil \frac{n}{2} - d \rceil$
6.) Proof:

$$\begin{aligned} T(n) &\leq c \cdot log_2(\lceil \frac{n}{2} - d \rceil) + 1 \\ &\leq c \cdot log_2(\frac{n}{2} - d + 1) + 1 \\ &= c \cdot log_2(\frac{n - 2b + 2}{2}) + 1 \\ &= c \cdot log_2(n - 2b + 2) - c + 1; c > 1 \\ &\leq c \cdot log_2(n - 2b + 2); b > 1 \\ &\leq c \cdot log_2(n - b) \end{aligned}$$

$$T(n) = 2 \cdot T(\lfloor \sqrt{n} \rfloor) + lg(n) \tag{25}$$

1.) Using a new variable $m = log_2(n)$, we get $T(2^m) = 2T(2^{m/2}) + m$
2.) Rename $T(2^m)$ to $V(m)$ to get new recurrence $V(m) = 2V(\frac{m}{2}) + m$.
3.) Prove that this is bounded by $O(m \cdot log_2(m))$
4.) Use $m = log_2(n)$ and get $O(log_2(n) \cdot log_2(log_2(n))$

4 Probabilistic analysis

Use indicator random variable to solve the following problems.

4.1 Sum of n dice

What is the expected value of the sum of n six sided dice throws.

Solution:

X_i - Dice lands on i

$$P(X_i = 1) = \frac{1}{6}$$

$$E(X) = E[\sum_{i=1}^{6} i \cdot X_i] =$$

$$= \sum_{i=1}^{6} i \cdot E[X_i] =$$

$$= \frac{1}{6} \sum_{i=1}^{6} i = 3.5$$

For n throws the expected value is $3.5n$.

4.2 Hat-check problem

Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

Solution:

X - Number of persons who get their hat back.

X_i - Person i gets their hat back

$$X = \sum_{i=1}^{n} X_i$$

$$P(X_i = 1) = \frac{1}{n} \Rightarrow E[X_i] = \frac{1}{n}$$

$$E[X] = E[\sum_{i=1}^{n} X_i] =$$

$$= \sum_{i=1}^{n} E[X_i] =$$

$$= \sum_{i=1}^{n} \frac{1}{n} =$$

$$= n\frac{1}{n} = 1$$

4.3 Inversions

Given an array of n randomly permutated distinct numbers marked A_i for $i = 1 \dots n$. If $i < j$ and $A_i > A_j$, then the pair (i, j) is called an inversion. Use indicator random variables to compute the expected number of inversions.

Solution:

X - Number of inversions in array A

X_{ij} - Elements at position i and j make an inversion

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

$$P(X_{ij} = 1) = \frac{1}{2} \Rightarrow E[X_{ij}] = \frac{1}{2}$$

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] =$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] =$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} =$$

$$= \frac{n(n-1)}{2} \frac{1}{2} = \frac{n(n-1)}{4}$$

4.4 Balls and bins

You are given n bins and n balls. You throw each ball into the bins. Assume each ball can equally likely fall into any bin.

Useful equations:

$p -$ probability of a single hit

k balls fall into bin i.

$$P(X_i = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

useful limit

$$\lim_{n \to \infty} (1 - \frac{1}{n})^n = \frac{1}{e}$$

a) What is the expected number of empty bins?
Solution:

X - Number of empty bins
X_i - Bin i is empty
$$X = \sum_{i=1}^{n} X_i$$
$$P(X_i = 1) = (1 - \frac{1}{n})^n \approx \frac{1}{e} = E[X_i]$$
$$E[X] = E[\sum_{i=1}^{n} X_i] =$$
$$= \sum_{i=1}^{n} E[X_i] =$$
$$\approx \sum_{i=1}^{n} \frac{1}{e} \approx$$
$$\approx \frac{n}{e}$$

b) What about bins with exactly one ball?
Solution:

X - Number of bins with exactly one ball
X_i - Bin i has exactly one ball
$$X = \sum_{i=1}^{n} X_i$$
$$P(X_i = 1) = n\frac{1}{n}(1 - \frac{1}{n})^{n-1} =$$
$$= (1 - \frac{1}{n})^{n-1} \approx \frac{n^2}{e(n-1)} \approx \frac{1}{e} \approx E[X_i]$$
$$E[X] = E[\sum_{i=1}^{n} X_i] =$$
$$= \sum_{i=1}^{n} E[X_i] =$$
$$\approx \sum_{i=1}^{n} \frac{1}{e} \approx$$
$$\approx \frac{n}{e}$$

c) What about bins with exactly two balls?
Solution:

X - Number of bins with exactly two balls

X_i - Bin i has exactly two balls

$$X = \sum_{i=1}^{n} X_i$$

$$P(X_i = 1) = \frac{n(n-1)}{2}(\frac{1}{n})^2(1 - \frac{1}{n})^{n-2} =$$

$$= \frac{n-1}{2n}(1 - \frac{1}{n})^{n-2} \approx \frac{1}{2e} \approx E[X_i]$$

$$E[X] = E[\sum_{i=1}^{n} X_i] =$$

$$= \sum_{i=1}^{n} E[X_i] =$$

$$\approx \sum_{i=1}^{n} \frac{1}{2e} \approx$$

$$\approx \frac{n}{2e}$$

d) What about bins with more than 1 ball? **Solution:**

X - Number of bins with more than one ball

X_i - Bin i has more than one ball

$$X = \sum_{i=1}^{n} X_i$$

We can use answer from a) and b)

$$P(X_i = 1) \approx 1 - \frac{1}{e} - \frac{1}{e} \approx \frac{e-2}{e} \approx E[X_i]$$

$$E[X] = E[\sum_{i=1}^{n} X_i] =$$

$$= \sum_{i=1}^{n} E[X_i] =$$

$$\approx \sum_{i=1}^{n} \frac{e-2}{e} \approx$$

$$\approx \frac{n(e-2)}{e} \approx 0.264n$$

4.5 Hire assistant

As in the case of hire assistant you heard of in lectures. What is the probability that the algorithm hires.

a) Exactly once.

Answer : $\frac{1}{n}$

b) Exactly n times.

Answer : $\frac{1}{n!}$

c) Exactly twice.

Solution:

Lets note that the first in line must be candidate X_i whose rank is not n since that would mean we only hire one candidate. Somewhere else the best candidate Y must be placed and all candidates between X_i and Y are worse than X_i. This means that Y must be picked before all candidates better than X_i. Number of these candidates is $n-1$.

X_i - Candidate i is picked first

$$\sum_{i=1}^{n-1} P(X_i)P(Y) =$$

$$= \sum_{i=1}^{n-1} \frac{1}{n}\frac{1}{n-i} =$$

$$= \frac{1}{n}H_{n-1}$$

4.6 Random generator

You are given a function BiasedRandom() which returns TRUE with probability p and false with probability $1-p$. $0 < p < 1$.

a) Construct a function UnbiasedRandom() that uses BiasedRandom() which must return TRUE with probability $\frac{1}{2}$ and FALSE with probability $\frac{1}{2}$.

Algorithm 21

```
1: function UnbiasedRandom
2:     while TRUE do
3:         x ← BiasedRandom()
4:         y ← BiasedRandom()
5:         if x ≠ y then
6:             return x
7:         end if
8:     end while
9: end function
```

b) Prove that your function works.
Solution :
Lets look at probabilities of two throws:

$$\begin{aligned}\text{TRUE, TRUE} &: p^2\\ \text{TRUE, FALSE} &: p(1-p)\\ \text{FALSE, TRUE} &: (1-p)p\\ \text{FALSE, FALSE} &: (1-p)^2\end{aligned}$$

The loop only ends in cases when $x \neq y$ and they both have same probability.
c) Analyse time complexity of UnbiasedRandom().
Solution :
The probability of ending the loop is $\frac{1}{2p(1-p)}$ which directly leads to an answer of $\Theta(\frac{1}{2p(1-p)})$.

5 Amortized analysis

5.1 Comparing methods

Perform a sequence of n operations on a data structure in which almost every operation has a constant cost $c_i = 1$. Exceptions are operations c_i where $i = 2^k$ and $k \in \mathbb{N}$, these operations costs $c_i = i$. Show that amortized cost per operation is constant using:

1. Aggregate method
2. Accounting method
3. Potential method

Solution 1: Aggregate method
We need the exact sum of all operations. We will make a first sum assuming all the operations are cheap (cost 1) and then add the cost of $\log(n)$ expensive operations.

$$\begin{aligned} c = \sum_{i=1}^{n} c_i = & \qquad (26)\\ = \sum_{i=1}^{n} 1 - \sum_{i=1}^{\log(n)} 1 + \sum_{i=1}^{\log(n)} 2^i = & \qquad (27)\\ = n - \log(n) + 2(n-1) \leq & \qquad \leq 3n = O(n) \qquad (28)\end{aligned}$$

We need $O(n)$ time for n operations meaning $O(1)$ per operation.
Solution 2: Accounting method
We have to show that we can choose a constant $\hat{c}_i$ such that $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$ will always hold. We can use the results from aggregate method and trivially show that this is true for $\hat{c}_i = 3$.

Solution 3: Potential method

We need to define D_i which is our data structure after i-th operation. $\Phi(D_i)$ is the potential of our data structure after i-th operation. It must hold t$\Phi(D_i) > \Phi(D_0)$, where $\Phi(D_0)$ is usually set to 0. We need to guess potential function $\Phi(D_i)$ and calculate amortized cost $\hat{c}_i$ of all operations. All $\hat{c}_i$ must be constants if we want to prove amortized constant time of operations.

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

$\Phi(D_i) = 2i - 2^{\lfloor \log_2(i) \rfloor + 1}$

Cheap operations:

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) =$

$= 1 + 2i - 2^{\lfloor \log_2(i) \rfloor + 1} - (2(i-1) - 2^{\lfloor \log_2(i-1)) \rfloor + 1}) =$

$\left(2^{\lfloor \log_2(i) \rfloor + 1} = 2^{\lfloor \log_2(i-1) \rfloor + 1}\right)$, since $i \neq 2^k$

$= 3$

Expensive operations:

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) =$

$= i + 2i - 2^{\lfloor \log_2(i) \rfloor + 1} - (2(i-1) - 2^{\lfloor \log_2(i-1)) \rfloor + 1}) =$

$\left(\frac{1}{2}2^{\lfloor \log_2(i) \rfloor + 1} = 2^{\lfloor \log_2(i-1) \rfloor + 1}\right)$, since $i = 2^k$

$= i + 2i - 2^{\lfloor \log_2(i) \rfloor + 1} - (2(i-1) - 2^{\lfloor \log_2(i-1)) \rfloor + 1}) =$

$= i + 2i - 2i - (2(i-1) - 2\frac{i}{2})$

$= 2$

5.2 Full analysis of dynamic tables

You are given a dynamic table as shown on lectures which doubles its size when it's full but can also contratct to save space. Such dynamic tables use both dynamic expansions and contractions. Let's define num_i as the number of elements in the table after i-th operation and size_i the size of dynamic table after i-th operation. Load factor is defined as $\alpha_i = \frac{\text{num}_i}{\text{size}_i}$.

1. Suppose you contract the table as soon as the load factor falls to $\alpha_i < \frac{1}{2}$. Show that in this case the worst time per operation is $O(n)$.

2. Find a loading factor α at which you need to halve the table so that cost per operation remains constant. Prove it using potential method.

Solution 1: Example

Lets assume that our table is currently empty with $\text{size}_i = n$. First lets make n INSERT operations, which all take $O(1)$ time. After this another insert takes $O(n)$ time, since the table expands. If we use DELETE now, the table with contract resulting in another $O(n)$ operation. Repeating INSERT and DELETE will now force the structure to a loop of $O(n)$ operations.

Solution 2: Full table analysis

It is easy to see that a value of $\alpha_i = \frac{1}{4}$ works.

The potential function we will be using is:

$$\Phi(D_i) = \begin{cases} 2\,\text{num}_i - \text{size}_i \ ; \ \alpha_i \geq \frac{1}{2} \\ \frac{\text{size}_i}{2} - \text{num}_i \ ; \ \alpha_i < \frac{1}{2} \end{cases}$$

First we have to notice that we have 8 cases for this problem. We need to analyse INSERT and DELETE. And see what happens when α_{i-1} is above or under $\frac{1}{2}$ and the same for α_i.

INSERT:	DELETE:
$\alpha_{i-1} \geq \frac{1}{2}$	$\alpha_{i-1} \geq \frac{1}{2}$
a) EXPANSION	e) $\alpha_i \geq \frac{1}{2}$
b) NO EXPANSION	f) $\alpha_i < \frac{1}{2}$
$\alpha_{i-1} < \frac{1}{2}$	$\alpha_{i-1} < \frac{1}{2}$
c) $\alpha_i < \frac{1}{2}$	g) CONTRACTION
d) $\alpha_i \geq \frac{1}{2}$	h) NO CONTRACTION

a) INSERT, $\alpha_{i-1} \geq \frac{1}{2}$, EXPANSION

$$\begin{aligned} c_i &= \text{num}_{i-1} + 1 \\ \text{num}_i &= \text{num}_{i-1} + 1 \\ \text{size}_i &= 2\,\text{size}_{i-1} \\ \text{num}_{i-1} &= \text{size}_{i-1} \\ \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= \text{num}_i + 2\,\text{num}_i - \text{size}_i - (2\,\text{num}_{i-1} - \text{size}_{i-1}) \\ &= \text{num}_{i-1} + 1 + 2(\text{num}_{i-1} + 1 - \text{size}_{i-1}) - (2\,\text{num}_{i-1} - \text{size}_{i-1}) \\ &= 3 + \text{num}_{i-1} - \text{size}_{i-1} \\ &= 3 \end{aligned}$$

b) INSERT, $\alpha_{i-1} \geq \frac{1}{2}$, NO EXPANSION

$$\begin{aligned} c_i &= 1 \\ \text{num}_i &= \text{num}_{i-1} + 1 \\ \text{size}_i &= \text{size}_{i-1} \\ \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 2\,\text{num}_i - \text{size}_i - (2\,\text{num}_{i-1} - \text{size}_{i-1}) \\ &= 1 + 2\,\text{num}_{i-1} + 2 - \text{size}_{i-1} - 2\,\text{num}_{i-1} + \text{size}_{i-1} \\ &= 3 \end{aligned}$$

c) INSERT, $\alpha_{i-1} < \frac{1}{2}$, $\alpha_i < \frac{1}{2}$

$$
\begin{aligned}
c_i &= 1 \\
\text{num}_i &= \text{num}_{i-1} + 1 \\
\text{size}_i &= \text{size}_{i-1} \\
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= 1 + \frac{\text{size}_i}{2} - \text{num}_i - (\frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}) \\
&= 1 + \frac{\text{size}_{i-1}}{2} - \text{num}_{i-1} - 1 - \frac{\text{size}_{i-1}}{2} + \text{num}_{i-1} \\
&= 0
\end{aligned}
$$

d) INSERT, $\alpha_{i-1} < \frac{1}{2}$, $\alpha_i \geq \frac{1}{2}$

$$
\begin{aligned}
c_i &= 1 \\
\text{num}_i &= \text{num}_{i-1} + 1 \\
\text{size}_i &= \text{size}_{i-1} \\
\alpha_i &= \frac{1}{2} = \frac{\text{num}_i}{\text{size}_i} \rightarrow \text{size}_i = 2\,\text{num}_i, \text{size}_{i-1} = 2\,\text{num}_{i-1} + 2 \\
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= 1 + 2\,\text{num}_i - \text{size}_i - (\frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}) \\
&= 1 + 2\,\text{num}_{i-1} + 2 - \text{size}_{i-1} - \frac{\text{size}_{i-1}}{2} + \text{num}_{i-1} \\
&= 3 + 3\,\text{num}_{i-1} - \frac{3}{2}\,\text{size}_{i-1} \\
&= 3 + 3\,\text{num}_{i-1} - \frac{3(2\,\text{num}_{i-1} + 2)}{2} \\
&= 3
\end{aligned}
$$

e) DELETE, $\alpha_{i-1} \geq \frac{1}{2}$, $\alpha_i \geq \frac{1}{2}$

$$
\begin{aligned}
c_i &= 1 \\
\text{num}_i &= \text{num}_{i-1} - 1 \\
\text{size}_i &= \text{size}_{i-1} \\
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= 1 + 2\,\text{num}_i - \text{size}_i - (2\,\text{num}_{i-1} - \text{size}_{i-1}) \\
&= 1 + 2\,\text{num}_{i-1} - 2 - \text{size}_{i-1} - 2\,\text{num}_{i-1} + \text{size}_{i-1} \\
&= -1
\end{aligned}
$$

f) DELETE, $\alpha_{i-1} \geq \frac{1}{2}$, $\alpha_i < \frac{1}{2}$

$$
\begin{aligned}
c_i &= 1 \\
\text{num}_i &= \text{num}_{i-1} - 1 \\
\text{size}_i &= \text{size}_{i-1} \\
\alpha_{i-1} &= \frac{1}{2} = \frac{\text{num}_{i-1}}{\text{size}_{i-1}} \rightarrow \text{size}_{i-1} = 2\,\text{num}_{i-1} \\
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= 1 + \frac{\text{size}_i}{2} - \text{num}_i - (2\,\text{num}_{i-1} - \text{size}_{i-1}) \\
&= 1 + \frac{\text{size}_{i-1}}{2} - \text{num}_{i-1} + 1 - 2\,\text{num}_{i-1} + \text{size}_{i-1} \\
&= 2 + \frac{3}{2}\,\text{size}_{i-1} - 3\,\text{num}_{i-1} \\
&= 2
\end{aligned}
$$

g) DELETE, $\alpha_{i-1} < \frac{1}{2}$, CONTRACTION

$$
\begin{aligned}
c_i &= \text{num}_{i-1} \\
\text{num}_i &= \text{num}_{i-1} - 1 \\
\text{size}_i &= \frac{\text{size}_{i-1}}{2} \\
\text{num}_i &= \frac{\text{size}_i}{2} \\
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= \text{num}_{i-1} + 2\,\text{num}_i - \text{size}_i - (\frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}) \\
&= \text{num}_{i-1} + 2\,\text{num}_i - \text{size}_i - (\text{size}_i - (\text{num}_i + 1)) \\
&= \text{num}_{i-1} + 2\,\text{num}_i - \text{size}_i - \text{size}_i + \text{num}_i + 1 \\
&= 1 + 4\,\text{num}_i - 2\,\text{size}_i \\
&= 0
\end{aligned}
$$

h) DELETE, $\alpha_{i-1} < \frac{1}{2}$, NO CONTRACTION

$$
\begin{aligned}
c_i &= 1 \\
\text{num}_i &= \text{num}_{i-1} - 1 \\
\text{size}_i &= \frac{\text{size}_{i-1}}{2} \\
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= 1 + \frac{\text{size}_i}{2} - \text{num}_i - (\frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}) \\
&= 1 + \frac{\text{size}_i}{2} - \text{num}_i - (\frac{\text{size}_i}{2} - \text{num}_i - 1) \\
&= \text{num}_{i-1} + 2\,\text{num}_i - \text{size}_i - \text{size}_i + \text{num}_i + 1 \\
&= 2
\end{aligned}
$$

6 Appendix

6.1 Simplified Masters

$$
\begin{aligned}
T(n) &= aT(\frac{n}{b}) + \Theta(n^d), \\
a &\geq 1, \\
b &> 1, \\
d &\geq 0.
\end{aligned}
$$

$$
\begin{aligned}
Case1 &: a > b^d \rightarrow T(n) = \Theta(n^{log_b a}) \\
Case2 &: a = b^d \rightarrow T(n) = \Theta(n^d log_b n) \\
Case3 &: a < b^d \rightarrow T(n) = \Theta(n^d)
\end{aligned}
$$

6.2 Masters

$$
\begin{aligned}
T(n) &= aT(\frac{n}{b}) + f(n), \\
a &\geq 1, \\
b &> 1.
\end{aligned}
$$

$$
\begin{aligned}
Case1 &: f(n) = O(n^{log_b a - \epsilon}) \rightarrow T(n) = \Theta(n^{log_b a}); \epsilon > 0 \\
Case2 &: f(n) = \Theta(n^{log_b a}) \rightarrow T(n) = \Theta(n^{log_b a} log(n)) \\
Case3 &: f(n) = \Omega(n^{log_b a + \epsilon}) \rightarrow T(n) = \Theta(f(n)); \epsilon > 0 \\
&\quad \text{in } af(\frac{n}{b}) \leq cf(n) \text{ for some } c < 1 \text{ and big enough n} \\
Case2ext &: f(n) = \Theta(n^{log_b a} log^k(n)) \rightarrow T(n) = \Theta(n^{log_b a} log^{k+1}(n))
\end{aligned}
$$

6.3 Akra-Bazzi

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n) + f(n) \text{ za } n > n_0,$$

$n_0 \geq \frac{1}{b_i}, n_0 \geq \frac{1}{1-b_i}$ for each i,

$a_i > 0$ for each i,

$0 < b_i < 1$ for each i,

$k \geq 1$,

$f(n)$ is non-negative function

$c_1 f(n) \leq f(u) \leq c_2 f(n)$, for each u satisfying condition: $b_i n \leq u \leq n$

$$T(n) = \Theta(n^p(1 + \int_1^n \frac{f(u)}{u^{p+1}} du))$$

we get p from:

$$\sum_{i=1}^{k} a_i b_i^p = 1$$

6.4 Extended Akra-Bazzi

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n + h_i(n)) + f(n) \text{ za } n > n_0,$$

all the conditions from Akra-Bazzi still hold, plus:

$|h_i(n)| = O(\frac{n}{log^2 n})$

6.5 Annihilators

Steps on solving linear recurrences.

- Write the recurrence in operator form
- Extract an annihilator for the recurrence
- Factor the annihilator (if necessary)
- Extract the generic solution from the annihilator
- Solve for coefficients using base cases (if known)

Operator	Definition
addition	$(f+g)(n) := f(n)+g(n)$
subtraction	$(f-g)(n) := f(n)-g(n)$
multiplication	$(\alpha \cdot f)(n) := \alpha \cdot (f(n))$
shift	$\boldsymbol{E}f(n) := f(n+1)$
k-fold shift	$\boldsymbol{E}^k f(n) := f(n+k)$
composition	$(\boldsymbol{X}+\boldsymbol{Y})f := \boldsymbol{X}f + \boldsymbol{Y}f$
	$(\boldsymbol{X}-\boldsymbol{Y})f := \boldsymbol{X}f - \boldsymbol{Y}f$
	$\boldsymbol{X}\boldsymbol{Y}f := \boldsymbol{X}(\boldsymbol{Y}f) = \boldsymbol{Y}(\boldsymbol{X}f)$
distribution	$\boldsymbol{X}(f+g) = \boldsymbol{X}f + \boldsymbol{X}g$

Operator	Functions annihilated	
$\boldsymbol{E}-1$	α	
$\boldsymbol{E}-a$	αa^n	
$(\boldsymbol{E}-a)(\boldsymbol{E}-b)$	$\alpha a^n + \beta b^n$	[if $a \neq b$]
$(\boldsymbol{E}-a_0)(\boldsymbol{E}-a_1)\cdots(\boldsymbol{E}-a_k)$	$\sum_{i=0}^{k} \alpha_i a_i^n$	[if a_i distinct]
$(\boldsymbol{E}-1)^2$	$\alpha n + \beta$	
$(\boldsymbol{E}-a)^2$	$(\alpha n+\beta)a^n$	
$(\boldsymbol{E}-a)^2(\boldsymbol{E}-b)$	$(\alpha n+\beta)a^n + \gamma b^n$	[if $a \neq b$]
$(\boldsymbol{E}-a)^d$	$(\sum_{i=0}^{d-1} \alpha_i n^i)a^n$	

If $\boldsymbol{X}$ annihilates f, then $\boldsymbol{X}$ also annihilates $\boldsymbol{E}f$.
If $\boldsymbol{X}$ annihilates both f and g, then $\boldsymbol{X}$ also annihilates $f \pm g$.
If $\boldsymbol{X}$ annihilates f, then $\boldsymbol{X}$ also annihilates αf, for any constant α.

If $\boldsymbol{X}$ annihilates f and $\boldsymbol{Y}$ annihilates g, then $\boldsymbol{X}\boldsymbol{Y}$ annihilates $f \pm g$.