CSCI-4470 HW-1

Due: Sep 10 2023

Homework 1

Jun Wang
ID: 811574679

Upload a soft copy of your answers **(single pdf file)** to the submission link on elc before the due date. The answers to the homework assignment should be your own individual work. Make sure to show all the work/steps in your answer to get full credit. Answers without steps or explanation will be given zero.

Extra credit: There is 5 percentage extra credit if you don't submit hand written homework including the figures. You can use latex or any other tool to write your homework. For figures you can use any drawing tool and include the figure as a jpeg or a png file in your latex file.

1. (10 points) Write a pseudo code that calculates frequency of a number in given array. Use the loop invariant technique i.e initialization, maintenance and termination steps and show that pseudo code calculates the frequency of a given number correctly.

HW Key:

```
int Frequency(A,num,n)
{
    c = 0;
    for i = 1 to n
        if(A[i] == num)
        {
            c = c + 1
        }
    return c
} // Frequency
```

Loop Invariant: At iteration `i`, `c` is the count of number num in the array `A[1, ..., i - 1]`

Initialization: Before the start the loop there are no elements to consider and hence count should be zero. `c` is also zero before the loop starts.

`Maintenance:` At the start of i^{th} iteration count variable `c` is the count of number in array `A[1, ..., i - 1]` . At the end of ith iteration count variable `c = c + 1` or `0` , `c = count(A[1, ..., i - 1]) + count(A[i])` where function count just counts the instances of variable num in the array. `c = count(A[1, ..., i - 1], A[i])` , `c = countA[1, ..., i]`

`Termination:` Loop terminates when `i = n + 1` at the point `c` is the count of number num in the array `A[1, ..., n]`

2. (10 points) Prove that $\lfloor \alpha n \rfloor + \lceil (1-\alpha)n \rceil = n$ for any integer n and real number α in the range $0 \leq \alpha \leq 1$).

HW-Key:

$$\begin{aligned} \lfloor \alpha n \rfloor + \lceil (1-\alpha)n \rceil &= \lfloor \alpha n \rfloor + \lceil n - \alpha n \rceil \\ &= \lfloor \alpha n \rfloor + n + \lceil -\alpha n \rceil \\ &= \lfloor \alpha n \rfloor + n - \lfloor \alpha n \rfloor \\ &= n \end{aligned}$$

The first line to the second line follows because `n` is an integer and $\lceil n + x \rceil = n + \lceil x \rceil$ for an integer `n` . The next line follows because $\lceil -x \rceil = -\lfloor x \rfloor$ for any `x` .

3. (5 points each) Use the Big-O definition to prove or disprove the following. You can assume that functions involved are asymptotically non negative functions.

- ***Big-O Notation (O)*: $f(n) = O(g(n))$ *if there exist constants* $c > 0$ *and* $n_0 \geq 0$ *such that* $f(n) \leq c \cdot g(n)$ *for all* $n \geq n_0$.**

HW-Key:

(a) Give $f(n) \in O(g(n))$

$$\begin{gathered} f(n) \leq cg(n) \qquad \text{eq}(1) \\ \text{for some } c \geq 0 \text{ and } n \geq n_0 \\ \text{Using eq}(1)\ g(n) \geq \frac{1}{cf(n)} \text{ for } n \geq n_0 \\ \text{hence } g(n) \in \Omega(f(n)) \text{ with constant } = \frac{1}{c} \text{ and } n \geq n_0 \end{gathered}$$

(b). $p(n) \in O(f(n))$ $\text{then}\, p(n)f(n) \neq O(f(n))$:

HW-Key:

$$\Rightarrow p(n) \leq cf(n) \qquad eq(1)$$
$$\text{for some } c \leq 0 \text{ and } n \geq n_0$$
$$\text{Multiplying } f(n) \text{ on both side of eq}(1) \text{ we have}$$
$$\text{then } p(n)f(n) \leq c(f(n))^2 \qquad eq(2)$$
$$\text{for some } c \geq 0 \text{ and } n \geq n_0$$
$$\text{eq}(2) \Rightarrow p(n)f(n) \in O(f(n))^2$$
$$\text{hence } p(n)f(n) \neq O(f(n))$$
$$p(n)f(n) \in O(f(n)) \text{ only when } f(n) \text{ is constant so it is not always true.}$$

(c). $\max\{f(n), g(n)\} \in O(f(n) + g(n))$:

HW-Key:

$\max\{f(n), g(n)\}$ is either $f(n)$ or $g(n)$
We know that $f(n) \leq f(n) + g(n)$ for $c = 1$ and some positive n
Similarly $g(n) \leq f(n) + g(n)$ for $c = 1$ and some positive n
Using the above three statements $max\{f(n), g(n)\} \in O(f(n) + g(n))$

(d). $n! \in O(n^n)$:

HW-Key:

$$\begin{aligned} n! &= n(n-1)(n-2)...1 \\ &\leq n \times n...n \\ n! &\leq n^n \end{aligned}$$

Hence $n! \in O(n^n)$ with $c = 1$ and $n \geq 0$

4. (5 points each) Prove the following:

(a). $f(n) = 4n^2 - 50n + 10 \in o(n^3)$:

- For little-o function, the "little-o notation" represents an asymptotic relationship between two functions, denoted as $f(x) = o(g(x))$. It is defined as: $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$, and indicates that $f(x)$ grows slower than $g(x)$ as x approaches infinity.
- By Applying the concept of ***little-o notation*** to prove that $f(n) \in o(n^3)$
  - $f(n) = 4n^2 - 50n + 10$, and $g(n) = n^3$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{4n^2 - 50n + 10}{n^3} = 0$$

$$\lim_{n\to\infty} \frac{4n^2}{n^3} - \frac{50n}{n^3} + \frac{10}{n^3} = \lim_{n\to\infty} \left(\frac{4}{n} - \frac{50}{n^2} + \frac{10}{n^3}\right) = 0$$

- By Applying the Infinity Property $\lim_{n\to\infty} \frac{c}{n^a} = 0$:
  - $\lim_{n\to\infty} \frac{4}{\infty} = 0$, $\lim_{n\to\infty} \frac{50}{\infty} = 0$, and $\lim_{n\to\infty} \frac{10}{\infty} = 0$
- **Conclusion**, By using the concept of "little-o notation". each term of $f(n)$ by n^3 and then took the limit as n approaches infinity, showing that the limit is 0, which confirms that $f(n) \in o(n^3)$.

(b). $f(n) = n^3 + 5n^2 - 5 \in \omega(n^2)$:

- For little-ω function, the "little-omega notation" represents an asymptotic relationship between two functions, denoted as $f(n) = \omega(g(n))$, If $f(n) > c \cdot g(n)$ for all constants $c > 0$ and for all sufficiently large n ($n \geq n_0$), then $f(n) \in \omega(g(n))$. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, and indicates that $f(n)$ grows faster than $g(n)$ as n approaches ∞.
- By Applying the concept of ***little-ω*** to prove that $f(n) \in \omega(n^2)$
  - $f(n) = n^3 + 5n^2 - 5$, and $g(n) = n^2$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3 + 5n^2 - 5}{n^2} = \infty$$

- Simplify the expression

$$\lim_{n\to\infty} \frac{n^3}{n^2} = \lim_{n\to\infty} \frac{5n^2}{n^2} = \lim_{n\to\infty} \frac{5}{n^2} = n + 5 - 0 = \infty$$

- **Conclusion**, By using the concept of "little-ω notation". The result for the limit function goes to ∞, therefore, $f(n) \in \omega(g(n))$

5. (5 points each) Use substitution method to find the asymptotic upper bound of the following recurrence relations.

HW-Key:

5[a] $T(n) = T(\frac{n}{2}) + n^3$:

We will assume solution of the form $T(n) \leq cn^3$

$$\begin{aligned} T(n) &\leq c(\frac{n}{2})^3 + n^3 \\ &= \frac{c(n^3)}{8} + n^3 \\ &= cn^3 - \frac{7cn^3}{8} + n^3 \\ &= cn^3 - n^3(\frac{7c}{8} - 1) \end{aligned}$$

Therefore as long as $\frac{7c}{8} - 1 \geq 0$ or $c \geq \frac{8}{7}$ and $n \geq 1$
$T(n) \in O(n^3)$

1. You're given the inequality:
   $\frac{7c}{8} - 1 \geq 0, 8 \cdot (\frac{7c}{8} - 1) \geq 8 \cdot 0$
   $7c - 8 \geq 0, 7c \geq 8$
   $7c \geq 8, \frac{7c}{7} \geq \frac{8}{7}, c \geq \frac{8}{7}$

HW-Key:

5[b] $f(n) = n^3 + 5n^2 - 5 \in \omega(n^2)$:

We will assume solution of the form $T(n) \leq cn\log(n)$

$$\begin{aligned} T(n) &\leq 3c(\frac{n}{3}\log(\frac{n}{3})) + n \\ &= cn\log(\frac{n}{3}) + n \\ &= cn\log(n) - cn\log(3) + n \\ &= cn\log(n) - n(c\log(3) - 1) \end{aligned}$$

Therefore as long as $c\log(3) - 1 \geq 0$ or $c \geq \frac{1}{\log(3)}$ and $n \geq 2$
assuming base of log is 2 then $T(n) \in O(n\log(n))$

HW-Key:

5[c] $T(n) = 3T(n-1) + 1$:

We will assume solution of the form $T(n) \leq c3^n$

$$\begin{aligned} T(n) &\leq 3c3^{n-1} + 1 \\ &= 3c\frac{3^n}{3} + 1 \\ &= c3^n + 1 \end{aligned}$$

We do not get the equation of the desired form there is an extra + 1 in our equation so we substract this term from our solution so our new solution is the form $T(n) \leq c3^n - d$

$$\begin{aligned} T(n) &\leq 3(c3^{n-1} - d) + 1 \\ &= 3c\frac{3^n}{3} - 3d + 1 \\ &= c3^n - 3d + 1 \\ &= c3^n - d - 2d + 1 \\ &= c3^n - d - (2d - 1) \end{aligned}$$

Therefore as long as $2d - 1 \geq 0$ or $d \geq \frac{1}{2}$ we also want first term to be positive and that happens when $c3^n - d \geq 0 \Rightarrow n \geq \log_3(\frac{d}{c})$, hence $T(n) \in O(3^n)$

6. (10 points) The solution to recurrence relation $T(n) = 8T(\frac{n}{2}) + n^2$ is $O(n^3)$. Show that using the substitution proof with the assumption $T(n) \leq cn^3$ fails. Subtract a lower order term from the assumption and show that $T(n) \in O(n^3)$

HW-Key:

$T(n) = 8T(\frac{n}{2}) + n^2$:

Using solution of the form $T(n) \leq cn^3$ we have

$$\begin{aligned} T(n) &\leq 8c(\frac{n}{2})^3 + n^2 \\ &= 8c\frac{n^3}{8} + n^2 \\ &= cn^3 + n^2 \end{aligned}$$

Now we will use solution of the form $T(n) \leq c_1n^3 - c_2n^2$

$$
\begin{aligned}
T(n) &\leq 8(c_1(\frac{n}{2})^3 - c_2(\frac{n}{2})^2) + n^2 \\
&= 8(c_1\frac{n^3}{8} - c_2\frac{n^2}{4}) + n^2 \\
&= c_1n^3 - 2c_2n^2 + n^2 \\
&= c_1n^3 - c_2n^2 - c_2n^2 + n^2 \\
&= c_1n^3 - c_2n^2 - n^2(c_2 - 1)
\end{aligned}
$$

Therefore as long as $c_2 - 1 \geq 0$ or $c_2 \geq 1$ we also want first term to be positive and that happens when $c_1n^3 - c_2n^2 \geq 0 \Rightarrow n \geq \frac{c_2}{c_1}$, therefore $T(n) \in O(n^3)$

7. (10 points) Use the recurrence tree method to solve the following recurrence relation $T(n) = 2T(\frac{n}{2}) + n^3$. Make sure to include a tree diagram and show all steps in your answer.

HW-Key:

Size of the problem at level $i = \frac{n}{2^i}$,
At the last level, the size of the problem is, $\frac{n}{2^i} = 1, i = \log(n)$
Therefore, there are log n levels in the tree.

Sum of nodes at level 0 are n^3

Sum of nodes at level 1 $= \frac{n^3}{2^3} + \frac{n^3}{2^3} = \frac{2n^3}{2^3} = \frac{n^3}{2^2}$

Sum of nodes at level 2 $= \frac{n^3}{4^3} + \frac{n^3}{4^3} + \frac{n^3}{4^3} + \frac{n^3}{4^3} = \frac{n^3}{4^2} = \frac{n^3}{2^4}$

Sum of nodes at level 3 $= \frac{n^3}{8^3} + \frac{n^3}{8^3} + \frac{n^3}{8^3} + \frac{n^3}{8^3} + \frac{n^3}{8^3} + \frac{n^3}{8^3} + \frac{n^3}{8^3} + \frac{n^3}{8^3} = \frac{n^3}{8^2} = \frac{n^3}{2^6}$

So, the sum of nodes at the level $i = \frac{n^3}{2^{2i}}$, The number of nodes at any level i is 2^i.

The number of nodes in the last level is $2^{\log(n)} = n$. The cost of each node is constant, so the total cost of last level is n.

Total cost of the tree is

$$\begin{aligned}\sum_{i=0}^{\log(n)} \frac{n^3}{2^{2i}} &= \sum_{i=0}^{\log(n)-1} \left(\frac{n^3}{2^{2i}}\right) + n \\ &= n^3 \sum_{i=0}^{\log(n)-1} \left(\frac{1}{2^{2i}}\right) + n \\ &\leq n^3 \sum_{i=0}^{\infty} \left(\frac{1}{2^{2i}}\right) \\ &= n^3 \left(\frac{1}{\left(1-\left(\frac{1}{4}\right)\right)}\right) + n \\ &= \frac{4n^3}{3} + n\end{aligned}$$

Therefore, $T(n) \in O(n^3)$

8. [a] (5 points each) Assume you have two fair dice. What is the expected value of a roll of these dice.

HW-Key:

X_1 : Outcome of first dice
X_2 : Outcome of second dice

$X = X_1 + X_2$

We have to calculate $E[X]$. $E[X] = E[X_1 + X_2] = E[X_1] + E[X_2]$
$E[X_1] =$ Expected value of outcome in dice 1.

There are six outcomes 1,2,3,4,5,6, each with probability $\frac{1}{6}$.

Therefore, $E[X_1] = \frac{1 \cdot 1}{6} + \frac{2 \cdot 1}{6} + \frac{3 \cdot 1}{6} + \frac{4 \cdot 1}{6} + \frac{5 \cdot 1}{6} + \frac{6 \cdot 1}{6}$
$= \frac{1+2+3+4+5+6}{6} = 2.34$

$E[X_1] = E[X_2]$.

Therefore, $E[X] = E[X_1] + E[X_2] = 2.34 + 2.34 = 4.68$

8. [b] Now assume that the two dice are biased and has the following probability distributions. What is the expected value of a roll of these dice?

Die 1	Value Probability	1 0.11	2 0.04	3 0.07	4 0.30	5 0.33	6 0.15
Die 2	Value Probability	1 0.17	2 0.28	3 0.21	4 0.07	5 0.03	6 0.24

HW-Key:

$E[X_1] = (1 \cdot 0.11) + (2 \cdot 0.04) + (3 \cdot 0.7) + (4 \cdot 0.3) + (5 \cdot 0.33) + (6 \cdot 0.15) = 6.04$

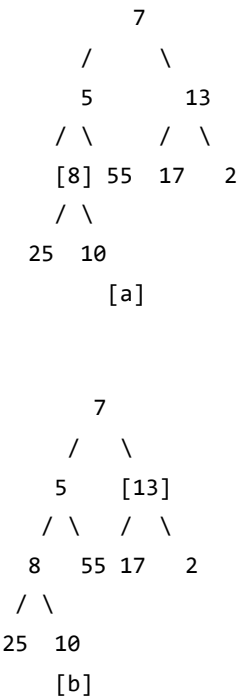
$E[X_2] = (1 \cdot 0.17) + (2 \cdot 0.28) + (3 \cdot 0.21) + (4 \cdot 0.7) + (5 \cdot 0.03) + (6 \cdot 0.24) = 3.23$

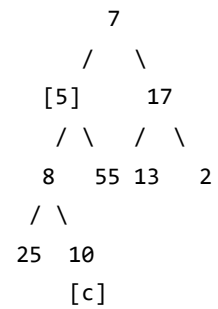
Therefore, $E[X] = E[X_1] + E[X_2] = 6.04 + 3.23 = 9.27$

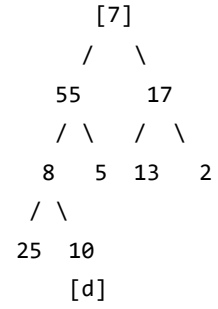
9. [a] (10 points) Similar to figure 6.3 illustrate the operation of BUILD-MAX-HEAP function (from the text book) on the following array $A =< 7, 5, 13, 8, 55, 17, 2, 25, 10 >$

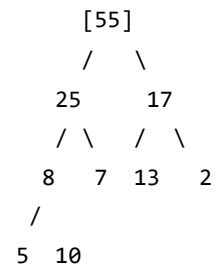
HW-Key: (a)

Start with the last non-leaf node in the heap i.e node 8 and call `MAX-HEAPIFY` . 25 is the bigger child of 8 and since 8 is smaller than 25 so we swap 8 and 25. The resultant heap shown in figure b. We repeat this for all the other non leaf nodes. The final array A is now a max-heap: `[55, 25, 17, 10, 5, 13, 2, 8, 7]` (figure e). It satisfies the max-heap property, where every parent node is greater than or equal to its child nodes.







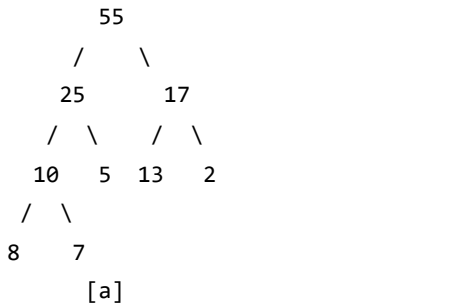


- $A =< 55, 25, 17, 10, 5, 13, 2, 8, 7 >$

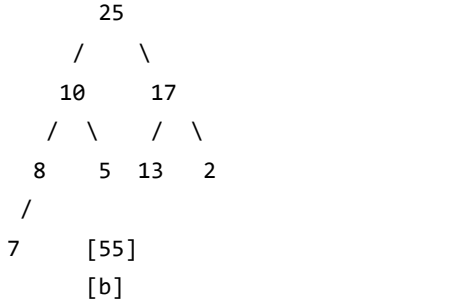
9. [b] (10 points) Use the heap constructed in part a and similar to figure 6.4 illustrate the operation of HEAPSORT function (from the text book) on the heap from part 9[a].

HW-Key: (b)
We can perform `HEAPSORT` on this max-heap as follows:

Initially, the max-heap contains the largest element at the root (index 0). To sort the array in the ascending order we will use the `HEAPSORT` algorithm given in the book. We'll repeatedly extract the maximum element from the heap and place it at the end of the array. We'll then adjust the heap to maintain its max-heap property.
8Figure b shows how the heap looks after 55 is sorted and removed from the heap.After all the swaps and `MAX-HEAPIFY` operations, the array is sorted in ascending order, and the largest element (55) is at the end: (figure i)

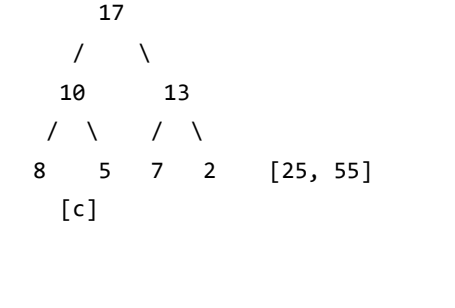


[a]



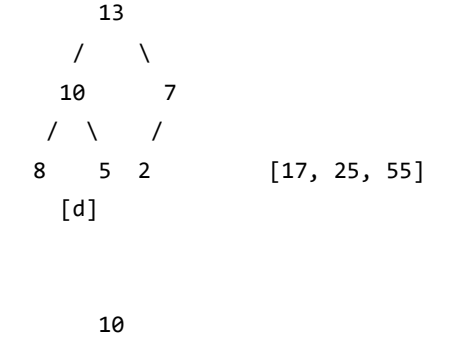
[55]

[b]



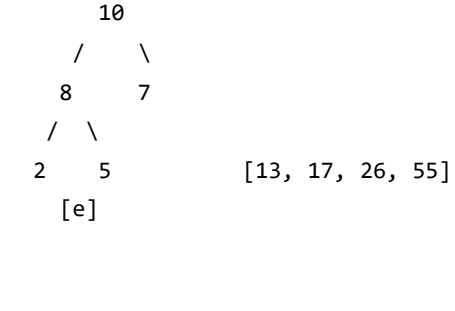
[25, 55]

[c]



[17, 25, 55]

[d]



[13, 17, 26, 55]

[e]

```
      8
    /    \
   5       7
  /
 2               [10, 13, 17, 25, 55]
  [f]


     7
   /   \
  5     2     [8, 10, 13, 17, 25, 55]
  [g]


  5
 /
2         [7, 8, 10, 13, 17, 25, 55]
  [h]


  2     [5, 7, 8, 10, 13, 17, 25, 55]
  [i]
```

- sorted $< 2, 5, 7, 8, 10, 13, 17, 25, 55 >$

10. (10 points) Assume you have k sorted lists containing a total of n items. You want to merge them into 1 sorted list. Give a pseudo code or steps of your algorithm. Using that pseudo code comment on the complexity of the algorithm. Your algorithm should use a heap to solve the problem.

HW-Key:

We will assume that each data has information about its list number as well. Lets say 20 is a number present in the list 3 so when we sort 20 we will know that it is from list 3 as well.

1. Take the largest number from each of the 'k' lists. Build a max-heap using these k numbers. The complexity of this operation is $O(k)$
2. We extract the max value from the heap and that is the sorted number(max of max). Complexity of this operation is $\log(k)$.
3. Lets say this maximum value belongs to list 2 so we go and add the next maximum value from list 2 to our current heap because second maximum can be from the same list. Adding a new number to the heap is going to be $\log(k)$.
4. Repeat the step 2 - 3 for all the remaining numbers in the lists.

5. Cost of step 1 is one time cost. Step 2 - 3 we do it for remaining $(n-k)$ elements so total cost is $(n-k)2\log(k)$ or $O(n\log(k))$

Before Homework

Question 1:

Pseudo-code:

```
function find_freq(arr, n)
    freq = 0
    for i = 0 to arr.length - 1
        if arr[i] == n
            freq = freq + 1
        end if
        // Loop body
    end for
    return freq
end function
```

By using the loop invariant technique to analysis loop invariants step by step.

Step 1: Initialization

- **Invariant**
  - At the starting of the function loop, the frequency of the number in the array from $index\ 0\ to\ -1\ is\ 0$, which is correctly represented by the `freq` variable.

Step 2: Maintenance

- **Invariant**
  - Before each iteration of the function, the `freq` variable correctly represents the frequency of the number in the array up to the current index. to maintain this by incrementing the `freq` variable each time to find the number in the array.
  - explicitly stating how the invariant is maintained across iterations by the for loop statement

Step 3: Termination

- **Invariant**

- At the end of the function loop, the `freq` variable correctly represents the frequency of the number in the entire array.

Conclusion, The loop invariant holds at initialization, is maintained during each iteration, and still holds at termination, the pseudo-code correctly calculates the frequency of a given number in the array.

Question 2:

By Using Properties of Floor and Ceiling Functions, e.g. $\lceil 3.14 \rceil = 4 \lfloor -2.7 \rfloor = -3$

Step 1: Define the Variables

- Assume a new variable $x = \alpha n$, $\alpha = \frac{x}{n}$,
- Find the expression for $1 - \alpha$, $1 - \alpha = 1 - \frac{x}{n}$
- Simplify the expression for $1 - \alpha$, $1 - \alpha = 1 - \frac{x}{n}$, to combine the terms $1 - \alpha = \frac{n}{n} - \frac{x}{n}$ so,
  [
  1 - \alpha = \frac{n - x}{n}
  $

Step 2: Express the Ceiling Function in Terms of the Floor Function

- By the Property that $\lceil y \rceil = -\lfloor -y \rfloor$, so we can express the ceiling function in terms of the floor function:
  - $\lceil (1 - \alpha)n \rceil = \lceil n - \alpha n \rceil = \lceil n - x \rceil = -\lfloor -(n - x) \rfloor$

Step 3: Substitute the Expression for the Ceiling Function into the Original, then to use the Floor Function Property

- $\lfloor \alpha n \rfloor + \lceil (1 - \alpha)n \rceil = \lfloor x \rfloor - \lfloor -(n - x) \rfloor$

The floor property that $\lfloor y \rfloor = y - \{y\}$, where $\{y\}$ is the fractional part of y.

- $\lfloor x \rfloor - \lfloor -(n - x) \rfloor = x - \{x\} - (-(n - x) - \{-(n - x)\})$

Step 5: Simplify the Expression

- Now we simplify the expression further:
  - $x - \{x\} + n - x - \{-(n - x)\} = n - \{x\} + \{n - x\}$
- Since $\{n - x\} = 1 - \{x\}$, we have:
  - $n - \{x\} + \{n - x\} = n - \{x\} + (1 - \{x\}) = n$

Conclusion. the expression $\lfloor \alpha n \rfloor + \lceil (1 - \alpha)n \rceil$ simplifies to n, proving the statement to be TRUE for all integers n and real numbers α in the range $0 < \alpha \leq 1$

3a.

(a) $f(n) \in O(g(n)) \Rightarrow g(n) \in \Omega(f(n))$:

- **Proof**: To prove the statement is TRUE,
  - By definition, If $f(n) \in O(g(n))$, there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. This implies that $g(n) \geq \frac{1}{c} \cdot f(n)$ for all $n \geq n_0$, and $g(n)$ is a lower bound for $f(n)$ which means $g(n) \in \Omega(f(n))$.

Proof: Assume for the sake of contradiction that $p(n) \in O(f(n))$ and $p(n)f(n) \in O(f(n))$.
證明: 為了證明這個命題，我們假設 $p(n) \in O(f(n))$ 和 $p(n)f(n) \in O(f(n))$。

- By definition, there exist constants $c_1, c_2 > 0$ and $n_0 \geq 0$ such that $p(n) \leq c_1 \cdot f(n)$ and $p(n)f(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$.
  根據定義，存在常數 $c_1, c_2 > 0$ 和 $n_0 \geq 0$ 使得對所有 $n \geq n_0$，有 $p(n) \leq c_1 \cdot f(n)$ 和 $p(n)f(n) \leq c_2 \cdot f(n)$。
- Multiplying the first inequality by $f(n)$, we have $p(n)f(n) \leq c_1 \cdot f(n)^2$.
  將第一個不等式乘以 $f(n)$，我們有 $p(n)f(n) \leq c_1 \cdot f(n)^2$。
- But we also have $p(n)f(n) \leq c_2 \cdot f(n)$, which is a contradiction unless $f(n) = 1$ or $c_2 \geq c_1 \cdot f(n)$, which cannot be guaranteed for all $f(n)$.
  但我們也有 $p(n)f(n) \leq c_2 \cdot f(n)$，這是一個矛盾，除非 $f(n) = 1$ 或 $c_2 \geq c_1 \cdot f(n)$，這不能保證對所有 $f(n)$ 都成立。
- Therefore, $p(n)f(n) \neq O(f(n))$ as the two conditions cannot hold simultaneously for all $f(n)$.
  因此，$p(n)f(n) \neq O(f(n))$，因為這兩個條件不能同時對所有 $f(n)$ 成立。

In the Big-O definition, to prove that $\max\{f(n), g(n)\} \in O(f(n) + g(n))$, you need to find constants c and n_0 such that for all $n > n_0$, $\max\{f(n), g(n)\} \leq c \cdot (f(n) + g(n))$.

In the counterexample you provided:

- $f(n) = n$ and $g(n) = n^2$.
- $\max\{f(n), g(n)\} = g(n) = n^2$.

You can choose $c = 1$ and $n_0 = 1$. For all $n > n_0$:

$$\max\{f(n), g(n)\} = n^2 \leq 1 \cdot (n + n^2) = n + n^2$$

So in this case, $\max\{f(n), g(n)\}$ is in $O(f(n) + g(n))$, and the statement is true for these functions.

To disprove the statement, you would need to find functions $f(n)$ and $g(n)$ for which no such constants c and n_0 exist.

在 Big-O 定義中，為了證明 $\max\{f(n), g(n)\} \in O(f(n) + g(n))$，你需要找到常數 c 和 n_0，使得所有 $n > n_0$，$\max\{f(n), g(n)\} \leq c \cdot (f(n) + g(n))$。

在你提供的反例中：

- $f(n) = n$ 和 $g(n) = n^2$。
- $\max\{f(n), g(n)\} = g(n) = n^2$。

你可以選擇 $c = 1$ 和 $n_0 = 1$。對於所有 $n > n_0$：

$$\max\{f(n), g(n)\} = n^2 \leq 1 \cdot (n + n^2) = n + n^2$$

所以在這種情況下，$\max\{f(n), g(n)\}$ 是 $O(f(n) + g(n))$，並且這些函數的語句是真實的。

要反駁該語句，你需要找到 $f(n)$ 和 $g(n)$ 的函數，對於這些函數不存在這樣的常數 c 和 n_0。

- **Disprove**: to prove the statement is FALSE.
  - Assume $f(n) = n$ and $g(n) = n^2$.
  - In this case, $\max\{f(n), g(n)\} = g(n) = n^2$.
    - Assume $n = 3$, then $f(3) = 3$, and $g(3) = 3^2 = 9$, the max value is $g(n) = n^2$.
  - $f(n) + g(n) = n + n^2$, and n^2 is not in $O(n + n^2)$ as n^2 will always dominate over $n + n^2$ for large n.
  - **Conclusion**, The disproof successfully finds a counterexample, showing that the statement is false.

3d:

- **Disprove:** to prove the statement is FALSE.
  - Two functions are given: $n!$ and n^n.
  - By Comparing the Growth Rates n^n grows faster than $n!$ as n becomes large.
  - Trying to find a Counterexample to disprove where $n!$ is not in $O(n^n)$. However, since n^n grows faster than $n!$
  - **Conclusion**, not possible to find out a counterexample and conclude that $n!$ is in $O(n^n)$, and a disproof solution is not possible.

4a:

[a] $f(n) = 4n^2 - 50n + 10 \in o(n^3)$:

- For little-o function, the “little-o notation” represents an asymptotic relationship between two functions, denoted as $f(x) = o(g(x))$. It is defined as: $\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$, and indicates that $f(x)$ grows slower than $g(x)$ as x approaches infinity.
- By Applying the concept of ***little-o notation*** to prove that $f(n) \in o(n^3)$
  - $f(n) = 4n^2 - 50n + 10$, and $g(n) = n^3$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{4n^2-50n+10}{n^3}=0$$

$$\lim_{n\to\infty}\frac{4n^2}{n^3}-\frac{50n}{n^3}+\frac{10}{n^3}=\lim_{n\to\infty}\left(\frac{4}{n}-\frac{50}{n^2}+\frac{10}{n^3}\right)=0$$

- By Applying the Infinity Property $\lim_{n\to\infty}\frac{c}{n^a}=0$:
  - $\lim_{n\to\infty}\frac{4}{\infty}=0$, $\lim_{n\to\infty}\frac{50}{\infty}=0$, and $\lim_{n\to\infty}\frac{10}{\infty}=0$
- **Conclusion**, By using the concept of "little-o notation". each term of $f(n)$ by n^3 and then took the limit as n approaches infinity, showing that the limit is 0, which confirms that $f(n)\in o(n^3)$.

[b] $f(n)=n^3+5n^2-5\in\omega(n^2)$:

- For little-ω function, the "little-omega notation" represents an asymptotic relationship between two functions, denoted as $f(n)=\omega(g(n))$, If $f(n)>c\cdot g(n)$ for all constants $c>0$ and for all sufficiently large n ($n\geq n_0$), then $f(n)\in\omega(g(n))$. $\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$, and indicates that $f(n)$ grows faster than $g(n)$ as n approaches ∞.
- By Applying the concept of ***little-ω*** to prove that $f(n)\in\omega(n^2)$
  - $f(n)=n^3+5n^2-5$, and $g(n)=n^2$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{n^3+5n^2-5}{n^2}=\infty$$

- Simplify the expression

$$\lim_{n\to\infty}\frac{n^3}{n^2}=\lim_{n\to\infty}\frac{5n^2}{n^2}=\lim_{n\to\infty}\frac{5}{n^2}=n+5-0=\infty$$

- **Conclusion**, By using the concept of "little-ω notation". The result for the limit function goes to ∞, therefore, $f(n)\in\omega(g(n))$

5. [a] $T(n)=T(\frac{n}{2})+n^3$:

- The recurrence relation of $T(n)=T(\frac{n}{2})+n^3$

$$T(n)=\begin{cases}1 & \text{if } n\leq 1\\ T\left(\frac{n}{2}\right)+n^3 & \text{if } n>1\end{cases}$$

For $n>1$, Assume $T(n)=O(n^3)$,

- $T(n)=T(\frac{n}{2})+n^3$ ...(1)

Find the solution holds for $T(\frac{n}{2})$, substitute n with $\frac{n}{2}$)

- $T(\frac{n}{2}) = T(\frac{\frac{n}{2}}{2} + (\frac{n}{2}))^3 = T(\frac{n}{2}) = T(\frac{n}{4} + (\frac{n}{2}))^3$

Assume $T(n) \leq cn^3$, then $T(\frac{n}{2}) \leq c(\frac{n}{2})^3$ ...(2)

Substitute (2) into (1), $T(n) \leq T(\frac{n}{2}) + n^3$

- $T(n) \leq \frac{cn^3}{8} + n^3$
- Simplify, $T(n) \leq n^3(\frac{c}{8} + 1)$
- Assume, $\frac{c}{8} + 1 = k$, where k is a constant
- $T(n) \leq kn^3$

Conclusion, Given condition that when $n \geq 2$, the term kn^3 is positive for all n, Therefore, the function $T(n) = O(n^3)$, derived kn^3 from previous steps, shows that it is positive for $n \geq 2$, $k > 0$ to get the upper bound $O(n^3)$

5. [b] $T(n) = 3T(\frac{n}{3}) + n$:

Step 1: Define the recurrence relation

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3T\left(\frac{n}{3}\right) + n & \text{if } n > 1 \end{cases}$$

Step 2: Assume $T(n) = O(n^k)$, where k is a constant

Step 3: Substitute the assumed solution into the recurrence relation

- $T(n) = 3T\left(\frac{n}{3}\right) + n$
- Assume $T(n) \leq cn^k$, then $T\left(\frac{n}{3}\right) \leq c\left(\frac{n}{3}\right)^k$

Step 4: Substitute the expression into the original recurrence relation

- $T(n) \leq 3\left(c\left(\frac{n}{3}\right)^k\right) + n$
- $T(n) \leq 3c\left(\frac{n^k}{3^k}\right) + n$

Simplify to get:

- $T(n) \leq 3c\left(\frac{n^k}{3^k}\right) + n$

Divide both the n^k and 3^k by 3^{k-1}

- $T(n) \leq 3c\left(\frac{n^k}{3^k} \cdot \frac{1}{3^{k-1}}\right) + n$
  - $3^k \cdot 3^{k-1} = 3^{2k-1}$
- $T(n) \leq 3c\left(\frac{n^k}{3^{2k-1}}\right) + n$

Step 5: Find the value of k that satisfies the equation such that the term n can be into $cn^k\left(\frac{1}{3^{k-1}}\right) = n$.

Divide the fraction inside the parenthesis by 3^{k-1}:

- $T(n) \leq 3c\left(\frac{n^k}{3^{2k-1}} \cdot \frac{1}{3^{k-1}}\right) + n$

Combine the Exponents In the expression, using the rule $a^m \cdot a^n = a^{m+n}$:

- $T(n) \leq 3c\left(\frac{n^k}{3^{3k-2}}\right) + n$
  - $3^{2k-1} \cdot 3^{k-1} = 3^{(2k-1)+(k-1)} = 3^{3k-2}$

Simplify the expression inside the parenthesis

- $T(n) \leq 3c\left(\frac{n^k}{3^{3k-2}}\right) + n = c\left(\frac{3n^k}{3^{3k-2}}\right) + n$

then,

- $T(n) \leq cn^k\left(\frac{1}{3^{3k-2-k}}\right) + n$
  - $3n^k = 3^1 \cdot n^k = 3^{1+k} = 3^{3k-2}$
  - $1 + k = 3k - 2, k = 1$
  - $3^{3k-2} - k = 3^{3k-2-k}$

Step 6: Substitute the New Exponent into expression

- $T(n) \leq cn^k\left(\frac{1}{3^{3k-2-k}}\right) + n$

By factoring out $3c$ to c to simplify the expression

- $T(n) \leq 3c\left(\frac{n^k}{3^{3k-2}}\right) + n$
- $T(n) \leq cn^k\left(\frac{1}{3^{3k-2-k}}\right) + n$
- $T(n) \leq cn^k\left(\frac{1}{3^{3k-2-k}}\right) + n = cn^k\left(\frac{1}{3^{2k-2}}\right) + n$

Assume $k = 2$,

- $T(n) \leq cn^2\left(\frac{1}{9}\right) + n$

Conclusion:, When $n \geq 2$, the term $cn^2\left(\frac{1}{9}\right)$ is positive for all n, the function $T(n) = O(n^2)$.

5. [c] $T(n) = 3T(n-1) + 1$:

Step 1: Define the recurrence relation

$$T(n) = \begin{cases} a & \text{if } n \leq 1, \text{ where } a \text{ is a constant} \\ 3T(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Step 2:, Assume $T(n) = 3T(n-1) + 1 = a_n 3^n + b_n$, Substitute the guess into the recurrence relation $T(n-1)$ with $a_{n-1}3^{n-1} + b_{n-1}$:

- $a_n 3^n + b_n = 3(a_{n-1}3^{n-1} + b_{n-1}) + 1$

Step 3: Simplify and Solve
Simplify and solve for a_n and b_n:

- $a_n 3^n + b_n = 3(a_{n-1}3^{n-1} + b_{n-1}) + 1$

Expand the Right Hand Side

- $a_n 3^n + b_n = 3a_{n-1}3^{n-1}3 + 3b_{n-1} + 1$

Simplify the equation $3^{n-1}3 = 3^{n-1+1} = 3^n$:

- $a_n 3^n + b_n = 3a_{n-1}3^n + 3b_{n-1} + 1]$

Step 4: Equate Coefficients like powers of 3 to find the values of a_n and b_n:

- $a_n = 3a_{n-1}$ (1) by equating Coefficients of 3^n
- $b_n = 3b_{n-1} + 1$ (2) by equating the constant terms

Step 5: Solve equations by using the equations from step 4:

- $a_n = 3a_{n-1}$ (1)
- $b_n = 3b_{n-1} + 1$ (2)

Solve for a_n equation (1), a_n can be expressed in terms of a_{n-1}, a_{n-2}, a_{n-3} and so on until we reach a_1:

- $a_n = 3a_{n-1} = 3(3a_{n-2}) = 3^2 a_{n-2} = \ldots = 3^{n-1}a_1$
- Therefore, $a_n = 3^{n-1}a_1$ (3)

Solve for b_n by using a similar approach for b_n from equation (2),

- $b_n = 3b_{n-1} + 1 = 3(3b_{n-2} + 1) + 1 = 3^2 b_{n-2} + 3 + 1 = \ldots = 3^{n-1}b_1 + 3^{n-2} + \ldots + 3 + 1$

Summing up the series

- $b_n = 3^{n-1}b_1 + \sum_{i=0}^{n-2} 3^i$ (4)

Step 6: Substitute a_n and b_n into the Guess

Substitute the expressions for a_n and b_n from equations (3) and (4) back into our guess for $T(n)$:

- $T(n) = a_n 3^n + b_n$
- $T(n) = (3^{n-1}a_1)3^n + \left(3^{n-1}b_1 + \sum_{i=0}^{n-2} 3^i\right)$

Step 7: Simplify to Find the General Solution

The recursive relations for a_n and b_n

- $a_n = 3a_{n-1}$
- $b_n = 3b_{n-1} + 1$

Finding General Expression of a_n

Using equation (1) to express $a * n$ in terms of $a * n - 1$, ..., a_1:

- $a_n = 3a_{n-1} = 3(3a_{n-2}) = 3^2a_{n-2} = \ldots = 3^{n-1}a_1$ (1)

Finding General Expression of b_n

To express $b * n$ in terms of $b * n - 1$, $b\_n - 2$, ... , b_1:

- $b_n = 3b_{n-1} + 1$

To substitute the expression for $b * n - 1$ with $b_n = 3b * n - 1 + 1$

- $b_n = 3^2b_{n-2} + 3 + 1 = 3^3b_{n-3} + 3^2 + 3 + 1 = \ldots = 3^{n-1}b_1 + 3^{n-2} + 3^{n-3} + \ldots + 3 + 1$

Summation for b_n:

- $b_n = 3b_{n-1} + 1$
- $b_n = 3(3b_{n-2} + 1) + 1$

Simplify the expression

- $b_n = 3^2b_{n-2} + 3 + 1$

Substituting the expression further for $b\_n - 2$:

- $b_n = 3^2(3b_{n-3} + 1) + 3 + 1$
- $b_n = 3^3b_{n-3} + 3^2 + 3 + 1$

The pattern for generalize this for k steps:

- $b_n = 3^kb_{n-k} + 3^{k-1} + 3^{k-2} + \ldots + 3^2 + 3 + 1$

Step 8: Sum up the Series

The first term as 1, common ratio as 3, and having k terms.

- $S_k = \frac{1-r^k}{1-r}$

Substituting $r = 3$ and $k = k$

- $S_k = \frac{1-3^k}{1-3}$
- $b_n = 3^kb_{n-k} + S_k$

Substituting the expression for S_k

- $b_n = 3^k b_{n-k} + \frac{1-3^k}{1-3}$

6. (10 points) The solution to recurrence relation $T(n) = 8T(\frac{n}{2}) + n^2$ is $O(n3)$. Show that using the substitution proof with the assumption $T(n) \leq cn^3$ fails. Subtract a lower order term from the assumption and show that $T(n) \in O(n^3)$

$T(n) = 8T(\frac{n}{2}) + n^2$:

- The given recurrence relation of the function is :

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 8T\left(\frac{n}{2}\right) + n^2 & \text{if } n > 1 \end{cases}$$

Assume a Solution Form,

- $T(n) \leq cn^3$, then $T(\frac{n}{2}) \leq c(\frac{n}{2})^3$

Substitute the Assumed Solution into the Recurrence

- $T(n) \leq 8c\left(\frac{n}{2}\right)^3 + n^2$

Simplify the Expression

- $T(n) \leq 8c\left(\frac{1}{8}\right) n^3 + n^2 = cn^3 + n^2$
- the assumption $T(n) \leq cn^3$ fails because of The additional n^2 term does not allow maintaining the inequality for $T(n)$.

Based on the Simplify the expression to adjust the new assumption to subtract a lower term dn^2 to facilitate the assumption.

- $T(n) \leq cn^3 - dn^2$, where c and d constants $c > 0$ and $d > 0$
- Replace n with $\frac{n}{2}$, $T(\frac{n}{2}) \leq c(\frac{n}{2})^3 - d(\frac{n}{2})^2$

Substitute $T(\frac{n}{2})$ into the Recurrence relation $T(n) = 8T(\frac{n}{2}) + n^2$

- $T(n) \leq 8\left(c\left(\frac{n}{2}\right)^3 - d\left(\frac{n}{2}\right)^2\right) + n^2$
- $T(n) \leq 8\left(c\left(\frac{n^3}{8}\right) - d\left(\frac{n^2}{4}\right)\right) + n^2$
- Simplify the Expression, $T(n) \leq cn^3 - 2dn^2 + n^2$

Find out the Constant $d = ?$

- $T(n) \leq cn^3 - 2dn^2 + n^2$
- $-2dn^2 + n^2 \leq 0$, then $n^2(-2d + 1) \leq 0$

- Rearrange the terms $n^2(1-2d) \leq 0$, for n^2 is always positive for $n > 0$
- Find constant d, $1-2d \leq 0$, gives the constant d, $1 \leq 2d$ or $d \geq \frac{1}{2}$

Conclusion, For a suitable choice of constant c, and $d = \frac{1}{2}$, the solution $T(n) \leq cn^3 - dn^2$ satisfies the recurrence function, that $T(n) \in O(n^3)$.

7. (10 points) Use the recurrence tree method to solve the following recurrence relation $T(n) = 2T(\frac{n}{2}) + n^3$. Make sure to include a tree diagram and show all steps in your answer.

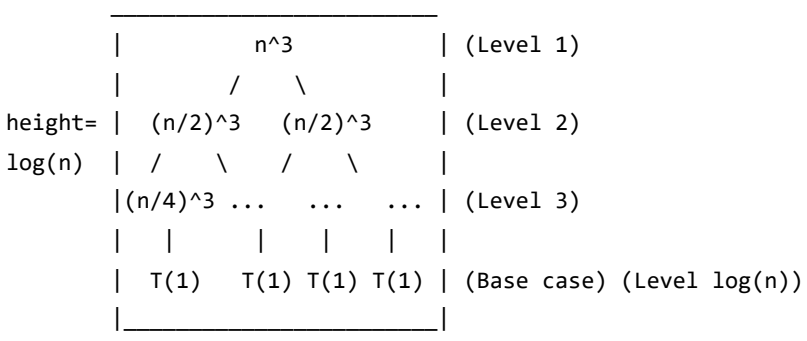
$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + n^3 & \text{if } n > 1 \end{cases}$$

Where:

- c is the base case constant time when n = 1
- The recursive case is when n > 1, dividing the problem into 2 subproblems of size n/2
- The base case T(1)
- The recursive case T(n) for n > 1

For the recurrence relation: $T(n) = 2T(\frac{n}{2}) + n^3$, When reached to LEVEL i, Get that $\frac{n}{2^i} = 1$, $2^i = n$, $i = \log(n)$

- The Recursive Tree Diagram



Calculating the Cost at Each Level of the tree in detail

- Level 0: n^3
- Level 1: $2 \cdot \left(\frac{n}{2}\right)^3 = \frac{1}{4}n^3$
  - $2 \cdot (\frac{n}{2})^3 = 2 \cdot (\frac{1}{8}n^3) = \frac{1}{4}n^3$
- Level 2: $4 \cdot \left(\frac{n}{4}\right)^3 = \frac{1}{4}n^3$
  - $4 \cdot (\frac{n}{4})^3 = 4 \cdot (\frac{1}{64}n^3) = \frac{1}{16}n^3$
- Level 3: $8 \cdot \left(\frac{n}{8}\right)^3 = \frac{1}{8}n^3$
  - $8 \cdot \left(\frac{n}{8}\right)^3 = 8 \cdot \left(\frac{1}{512}n^3\right) = \frac{1}{64}n^3$

- …
- Level i: $2^i \cdot \left(\frac{n}{2^i}\right)^3 = n^3$

Finding the Height of the Tree, to determine the levels until it reaches to base case that $T(1)$, $n = c$. Dividing n by 2 for each step $\frac{n}{2^h} = c$

- Solving for height h, $n = 2^h c$, then $2^h = \frac{n}{c}$
- Applying Logarithm for both side $\log(\frac{n}{c}) = \log(2^h)$
  - $\log(2^h) = h$, log_2 cancelled with 2,
- For h: $h = \log_2(\frac{n}{c})$
  -

The Height of the Tree is given by:

$$h = \log_2\left(\frac{n}{c}\right)$$

Sum up the total cost:

- Cost at level $i = 2^i \cdot (\frac{n}{2^i})^3$
- The total cost $T(n)$ is given by the sum of the costs from 0 to h, where h is the height of the recursive tree.
  - $T(n) = \sum_{i=0}^{h} 2^i \cdot (\frac{n}{2^i})^3$
  - $T(n) = \sum_{i=0}^{h} 2^i \cdot (\frac{1}{2^{3i}})n^3$
  - Simplify, $T(n) = n^3 \sum_{i=0}^{h} 2^i \cdot (\frac{1}{2^{3i}})$
  - Bring out Constants $T(n) = n^3 \sum_{i=0}^{h} (\frac{2^i}{2^{3i}})$
  - Simplify $T(n) = n^3 \sum_{i=0}^{h} (\frac{1}{2^{2i}})$

Geometric Series from the Total cost, the first term as 1 and the common ratio as $\frac{1}{4}$. We can find the sum of this series using the formula for the sum of a geometric series:

- $S_n = a + ar + ar^2 + ar^3 + ... + ar^n = n^3 \sum_{i=0}^{h} (\frac{1}{2^{2i}})$
- $T(n) = n^3(1 + \frac{1}{4} + (\frac{1}{4})^2 + (\frac{1}{4})^3 + ... + (\frac{1}{4})^h)$

Where:

- S_n is the sum of the series
- a is the first term of the series
- r is the common ratio of the series
- n is the number of terms in the series

Substitute the values that contained,

$$S_n = \frac{a(1 - r^n)}{1 - r}$$

$$T(n) = n^3 \cdot \frac{1(1 - (\frac{1}{4})^{h+1})}{1 - \frac{1}{4}}$$

8. [a] (5 points each) Assume you have two fair dice. What is the expected value of a roll of these dice.

Step 1: Define the Random Variable

- Random Variable X: The sum of the numbers obtained form rolling two fair dice

Step 2: The Sample Space and Probabilities

- **Sample Space,** S: $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- A total number of outcomes when rolling two dice(Each die has 6 faces) $6 \times 6 = 36$, then get the Probabilities are
  - $P(X) = \frac{number\ of\ outcomes}{Total\ number\ of\ outcomes}$
  - $P(2) = \frac{1}{36}, P(3) = \frac{2}{36}, P(4) = \frac{3}{36}$
  - $P(5) = \frac{4}{36}, P(6) = \frac{5}{36}, P(7) = \frac{6}{36}$
  - $P(8) = \frac{5}{36}, P(9) = \frac{4}{36}, P(10) = \frac{3}{36}$
  - $P(11) = \frac{2}{36}, P(12) = \frac{1}{36}$

Step 3: Define the Value Function

- **Value Function,** $v()$**, or The cost** x: The value function in this context is simply the identity function, i.e., $v(x) = x$, where x is any sum obtained from rolling the two dice.
  - $v(2) = 2, v(3) = 3, v(4) = 4, v(5) = 5, v(6) = 6$
  - $v(7) = 7, v(8) = 8, v(9) = 9, v(10) = 10, v(11) = 11$
  - $v(12) = 12$

Step 4: Calculate the Expected Value by the formula

- $$E[X] = \sum_{A \in S} (v(A) \cdot P(A))$$
- $$E[X] = (2 \cdot P(X = 2)) + (3 \cdot P(X = 3)) + (4 \cdot P(X = 4)) + \ldots + (12 \cdot P(X = 12))$$

Substitute the probabilities we calculated in step 2:

$$E[X] = (2 \cdot \frac{1}{36}) + (3 \cdot \frac{2}{36}) + (4 \cdot \frac{3}{36}) + \ldots + (12 \cdot \frac{1}{36})$$

Sum up these values to find the expected value:

$$E[X] = \frac{1}{36}((2 \cdot 1) + (3 \cdot 2) + (4 \cdot 3) + (5 \cdot 4) + (6 \cdot 5)$$
$$+(7 \cdot 6) + (8 \cdot 5) + (9 \cdot 4) + (10 \cdot 3) + (11 \cdot 2) + (12 \cdot 1) =$$

$$E[X] = \frac{1}{36}(290) = \frac{290}{36} \approx 8.0556$$

Conclusion, the expected value of a roll of the two fair dice is **approximately 8.0556.**

8. [b] Now assume that the two dice are biased and has the following probability distributions. What is the expected value of a roll of these dice?

Die 1	Value Probability	1 0.11	2 0.04	3 0.07	4 0.30	5 0.33	6 0.15
Die 2	Value Probability	1 0.17	2 0.28	3 0.21	4 0.07	5 0.03	6 0.24

Step 1: Define the Random Variable

- **Random Variable,** X: The sum of the values obtained from rolling the **two biased dice**

Step 2: Identify the Sample Space and Probabilities

- **Sample Space,** S: $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- **The probabilities** for each possible outcome when rolling the two biased dice
  - $P(X = 2) = P(\text{Die 1 rolls 1}) \cdot P(\text{Die 2 rolls 1}) = 0.11 \cdot 0.17$
  - $P(X = 3) = P(\text{Die 1 rolls 1}) \cdot P(\text{Die 2 rolls 2}) + P(\text{Die 1 rolls 2}) \cdot P(\text{Die 2 rolls 1}) = (0.11 \cdot 0.28) + (0.04 \cdot 0.17)$
  - $P(X = 4) = P(\text{Die 1 rolls 1}) \cdot P(\text{Die 2 rolls 3}) + P(\text{Die 1 rolls 2}) \cdot P(\text{Die 2 rolls 2}) + P(\text{Die 1 rolls 3}) \cdot P(\text{Die 2 rolls 1}) = (0.11 \cdot 0.21) + (0.04 \cdot 0.28) + (0.07 \cdot 0.17)$
  - $P(X = 5) = P(\text{Die 1 rolls 1}) \cdot P(\text{Die 2 rolls 4}) + P(\text{Die 1 rolls 2}) \cdot P(\text{Die 2 rolls 3}) + P(\text{Die 1 rolls 3}) \cdot P(\text{Die 2 rolls 2}) + P(\text{Die 1 rolls 4}) \cdot P(\text{Die 2 rolls 1})$
    - $P(X = 5) = (0.11 \cdot 0.07) + (0.04 \cdot 0.21) + (0.07 \cdot 0.28) + (0.30 \cdot 0.17)$
    - $P(X = 5) = (0.0077) + (0.0084) + (0.0196) + (0.0510) = 0.0867$
  - ...
  - To repeat this process for $P(X = 6)$, $P(X = 7)$, ..., $P(X = 12)$ to find the probabilities for all outcomes.

X	$P(X)$
2	$0.11 \cdot 0.17 = 0.0187$
3	$(0.11 \cdot 0.28) + (0.04 \cdot 0.17) = 0.0308 + 0.0068 = 0.0376$
4	$(0.11 \cdot 0.21) + (0.04 \cdot 0.28) + (0.07 \cdot 0.17) = 0.0231 + 0.0112 + 0.0119 = 0.0462$
5	$(0.11 \cdot 0.07) + (0.04 \cdot 0.21) + (0.07 \cdot 0.28) + (0.30 \cdot 0.17) = 0.0077 + 0.0084 + 0.0196 + 0.0510 = 0.0867$
6	$(0.11 \cdot 0.03) + (0.04 \cdot 0.07) + (0.07 \cdot 0.21) + (0.30 \cdot 0.28) + (0.33 \cdot 0.17) = 0.0033 + 0.0028 + 0.0147 + 0.0840 + 0.0561 = 0.1609$
7	$(0.11 \cdot 0.24) + (0.04 \cdot 0.03) + (0.07 \cdot 0.07) + (0.30 \cdot 0.21) + (0.33 \cdot 0.28) + (0.15 \cdot 0.17) = 0.0264 + 0.0012 + 0.0049 + 0.0630 + 0.0924 + 0.0255 = 0.2134$
8	$(0.04 \cdot 0.24) + (0.07 \cdot 0.03) + (0.30 \cdot 0.07) + (0.33 \cdot 0.21) + (0.15 \cdot 0.28) = 0.0096 + 0.0021 + 0.0210 + 0.0693 + 0.0420 = 0.1440$
9	$(0.07 \cdot 0.24) + (0.30 \cdot 0.03) + (0.33 \cdot 0.07) + (0.15 \cdot 0.21) = 0.0168 + 0.0090 + 0.0231 + 0.0315 = 0.0804$
10	$(0.30 \cdot 0.24) + (0.33 \cdot 0.03) + (0.15 \cdot 0.07) = 0.0720 + 0.0099 + 0.0105 = 0.0924$
11	$(0.33 \cdot 0.24) + (0.15 \cdot 0.03) = 0.0792 + 0.0045 = 0.0837$
12	$(0.15 \cdot 0.24) = 0.0360$

Step 3: Define the Value Function

- **Value Function,** $v()$:
  - $v(2) = 2, v(3) = 3, v(4) = 4, v(5) = 5, v(6) = 6$
  - $v(7) = 7, v(8) = 8, v(9) = 9, v(10) = 10, v(11) = 11$
  - $v(12) = 12$

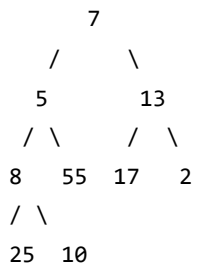
Step 4: Calculate the Expected Value by the formula

- $E[X] = \sum_{A \in S}(v(A) \cdot P(A))$
- $E[X] = \sum_{i=2}^{12}(i \cdot P(X = i))$
- $E[X] = (2 \cdot P(X = 2)) + (3 \cdot P(X = 3)) + (4 \cdot P(X = 4)) + \ldots + (12 \cdot P(X = 12))$
- $E[X] = (2 \cdot 0.0187) + (3 \cdot 0.0376) + (4 \cdot 0.0462) + (5 \cdot 0.0867) + (6 \cdot 0.1609) + (7 \cdot 0.2134) + (8 \cdot 0.1440) + (9 \cdot 0.0804) + (10 \cdot 0.0924) + (11 \cdot 0.0837) + (12 \cdot 0.0360)$

Step 5: Result

- $E[X] = 0.0374 + 0.1128 + 0.1848 + 0.4335 + 0.9654 + 1.4938 + 1.1520 + 0.7236 + 0.9240 + 0.9207 + 0.4320 = 7.3800$
- **Expected Value**: 7.3800
- Therefore, the expected value of a roll of these biased dice is approximately 7.3800.

9. [a] (10 points) Similar to figure 6.3 illustrate the operation of BUILD-MAX-HEAP function (from the text book) on the following array $A =< 7, 5, 13, 8, 55, 17, 2, 25, 10 >$



Step 1: Identifying the nodes with children by following approach

- The root node is at index 0.
- The left child of a node at index i is at index $2i + 1$.
- The right child of a node at index i is at index $2i + 2$.

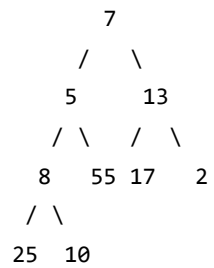
$A =< 7, 5, 13, 8, 55, 17, 2, 25, 10 >$, The array has 9 elements, and the last index is 8.

- Index 0 (value 7) has children at indices 1 and 2.
- Index 1 (value 5) has children at indices 3 and 4.
- Index 2 (value 13) has children at indices 5 and 6.
- Index 3 (value 8) has children at indices 7 and 8.

Therefore, the nodes with children are at **indices 0, 1, 2, and 3.**

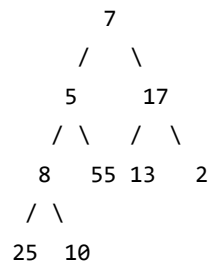
Step 2: Illustrating the operation with a diagram

- To illustrate the operation of the BUILD-MAX-HEAP function on the array



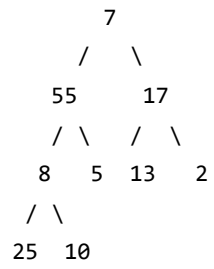
Step 2.1: Starting with the last node that has children (**indices 0, 1, 2, and 3.**) which is 2-indexed, and 3-indexed

- index 2 = 13, index 3 = 5, to compare their child nodes, switcher 2-indexed with child node that has value 17



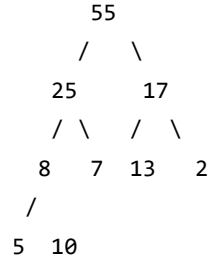
Step 2.2: Moving to the previous node

- Switch 3-indexed with child node which has the value 55.



Step 2.3: Moving to the root node

Finally, to move the root node which has the value 7 and swap it with the largest child to maintain the max heap property.

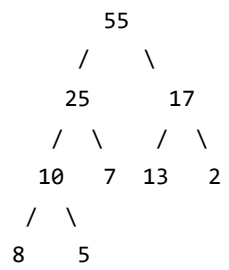


- the array represents a max heap. The array representation of this heap is:

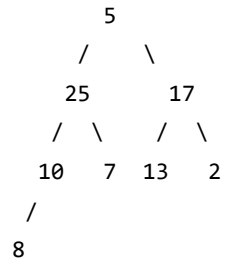
- $A =< 55, 25, 17, 8, 7, 13, 2, 5, 10 >$

9. [b] (10 points) Use the heap constructed in part a and similar to figure 6.4 illustrate the operation of HEAPSORT function (from the text book) on the heap from part 9[a].

The MAX HEAP of 9[a]:

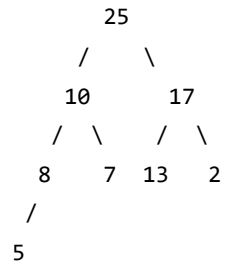


Step 1: Swap the Root (55) with the Last Element (5) and Remove the Last Element (55) from the heap.



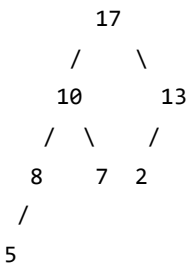
Step 2: Max Heapify

- To maintain the max heap property by performing max heapify on the root node.
  - 25 becomes the root, 5 goes to the last element

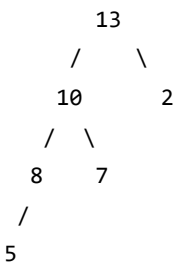


Step 3: Repeat step 1 and step 2 until the heap size reduces to 1.

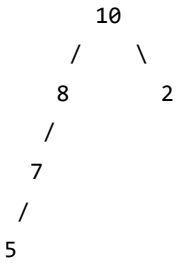
- switch the root (25) and the last element (5), remove the last element (25)
- to maintain the max heap property 17 becomes root, 5 goes the last element



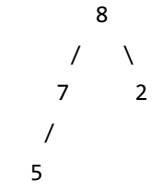
- repeat step 1 and step 2
  - switch the root (17) with the last element (5), remove the last element (17)
  - to maintain the max heap property 13 becomes the root, 5 goes the last elements



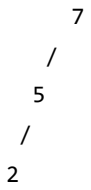
- repeat step 1 and step 2
  - switch the root (13) with the last element (5), remove the last element (13)
  - to maintain the max heap property 10 becomes the root, 5 goes the last elements



- repeat step 1 and step 2
  - switch the root (10) with the last element (5), remove the last element (10)
  - to maintain the max heap property 8 becomes the root, 5 goes the last elements



- repeat step 1 and step 2
  - switch the root (8) with the last element (5), remove the last element (8)
  - to maintain the max heap property 7 becomes the root, 5 goes the last elements



- repeat step 1 and step 2
  - switch the root (7) with the last element (5), remove the last element (7)
  - to maintain the max heap property 5 becomes the root, 5 goes the last elements

- repeat step 1 and step 2
  - switch the root (5) with the last element (2), remove the last element (5)
  - to maintain the max heap property 2 becomes the root, no child node remained.

Step 4: The Sorted Array

- the sorted array that collected the removed elements in each step:
- sorted $< 2, 5, 7, 8, 10, 13, 17, 25, 55 >$

10. (10 points) Assume you have k sorted lists containing a total of n items. You want to merge them into 1 sorted list. Give a pseudo code or steps of your algorithm. Using that pseudo code comment on the complexity of the algorithm. Your algorithm should use a heap to solve the problem.

Algorithm of k sorted lists to merge into 1 sorted list by a min-heap structure

Step 1: Create a Min-Heap (H):

- **1.1** Initialize a min-heap structure, denoted as H.
- **1.2** For each of the k sorted lists, insert the first element into H, setting it as the key for that list. This is an $O(1)$ operation for each list.

Step 2: Initialize Output Array (F) and Index (i):

- **2.1** Create an empty array F with a size of N to store the merged output.
- **2.2** Initialize an index i to 0, which will be used to track the current position in the output array.

Step 3: Merge Lists Using Min-Heap:

- **3.1** Enter a while loop that continues as long as H is non-empty.
- **3.2** Inside the loop, identify the list L at the root of H.
- **3.3** Extract the minimum number x from L and assign it to $F[i]$. Then, increment i by 1.
- **3.4** If L is empty after extracting x, perform an extract-min operation on H to remove L from H.
- **3.5** If L is not empty, update the key of L in H to the new minimum element of L and then call Heapify on H to restore the min-heap property.

Correctness:

- **4.1** The algorithm is correct because at each iteration, the minimum element among all uninserted elements is added to F.
- **4.2** This is ensured by the min-heap property, which always brings the list with the smallest uninserted element to the root.

Running Time:

- **5.1** Building the heap takes $O(K)$ time since each insertion is $O(1)$ and we have K insertions.
- **5.2** Each iteration in the while loop involves either an extract-min or a heapify operation, both taking $O(\log K)$ time. Since we have N iterations, this part takes $O(N \log K)$ time.
- **5.3** The total time complexity is $O(K + N \log K)$. Given $K \leq N$, it can also be expressed as $O(N \log K)$.

Conclusion:

- **6.1** The algorithm successfully merges k sorted lists into one sorted list using a min-heap, with a time complexity of $O(N \log K)$.
- **6.2** This ensures efficiency even for a large number of input lists and elements.

```
Algorithm MergeKSortedLists
Input: Array of K sorted lists containing a total of N elements
Output: A single sorted list containing all N elements

1. Initialize a min-heap, H
2. For each list in the input array:
   2.1. Insert the first element of the list into H, associating it with its originating list
3. Initialize an empty array F of size N to store the final sorted list
4. Initialize an index variable i to 0
5. While H is not empty:
   5.1. Identify the list, L, associated with the root of H
   5.2. Extract the minimum element, x, from L
   5.3. Set F[i] to x
   5.4. Increment i by 1
   5.5. If L is now empty:
       5.5.1. Extract the minimum from H (removing L's association)
   5.6. Else:
       5.6.1. Update the key of L in H to the new minimum element of L
       5.6.2. Call Heapify on H to restore the min-heap property
6. Return F
```