dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

CSCI 570 Homework 1

1. **SOLUTION:**

The given terms can be simplified as follows:

- $\sqrt{n}^{\sqrt{n}} = n^{\frac{\sqrt{n}}{2}}$
- $n^{\log n}$
- $n\ log\ (\ log\ n)$
- $n^{\frac{1}{\log n}} = 2$
- $2^{\log n} = n$
- $\log^2 n = (\log n).(\log n)$
- $(\log n)^{\sqrt{\log n}}$

- We know that function that grows the slowest is a constant which makes $n^{\frac{1}{\log n}}$ the first in the order.
- Next, we have $\log^2 n$ followed by $(\log n)^{\sqrt{\log n}}$ because in this hierarchy we have logarithmic functions after constants. Along with that we know that $(\log n)^{\sqrt{\log n}}$ grows faster than $\log^2 n$ as it is exponential in nature.
- Comparing $(\log n)^{\sqrt{\log n}}$ and $2^{\log n}$ which is 'n' we can take log on both sides and get terms $\sqrt{\log n}.\log(\log n)$ and $\log n$
  - Dividing by $\sqrt{\log n}$ on both sides, we get terms $\log(\log n)$ and $\sqrt{\log n}$
  - If we substitute log n by a variable 'x' then we can compare log x and $\sqrt{x}$ with the result of $\sqrt{x}$ growing faster than log x. Hence $2^{\log n} > (\log n)^{\sqrt{\log n}}$
- After this we get $2^{\log n}$ which is equal to *n* followed by $n\ log\ (\ log\ n\ )$
- We now need to compare $\sqrt{n}^{\sqrt{n}}$ which can be written as $n^{\frac{\sqrt{n}}{2}}$ and $n^{\log n}$
  - Assuming $n^{\frac{\sqrt{n}}{2}} \geq n^{\log n}$
  - Taking $\log_n(\ )$ on both the sides we get

$$\frac{\sqrt{n}}{2} \geq \log n$$

- As we know that $\frac{\sqrt{n}}{2}$ grows faster than $\log n$ we can conclude that $n^{\frac{\sqrt{n}}{2}} \geq n^{\log n}$ is true

Hence the order is

$$\boldsymbol{n^{\frac{1}{\log n}},\ \log^2 n,\ (\log n)^{\sqrt{\log n}},\ 2^{\log n}, n\log(\ log\ n),\ n^{\log n}, \sqrt{n}^{\sqrt{n}}}$$

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

2. PROOF: by induction

Base Case:

- Given that $F_n = \frac{(a^n - b^n)}{\sqrt{5}}$

  Where $a = \frac{1+\sqrt{5}}{2}$ and $b = \frac{1-\sqrt{5}}{2}$
- Consider for n = 3,

$$F_3 = \frac{(a^3 - b^3)}{\sqrt{5}} = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^3 - \left(\frac{1-\sqrt{5}}{2}\right)^3}{\sqrt{5}} = \frac{\frac{1+3\sqrt{5}+3*5+5\sqrt{5}}{8} - \frac{1-3\sqrt{5}+3*5-5\sqrt{5}}{8}}{\sqrt{5}} = \frac{6\sqrt{5}+10\sqrt{5}}{8\sqrt{5}} = 2$$

Induction Hypothesis:

- Assuming $P(k)$ is true for some k: $F_k = F_{k-1} + F_{k-2}$ for $k \geq 3$

Induction Step:

- We need to prove that $P(k+1)$ is also true:

$$F_{k-1} + F_k = \frac{(a^{k-1} - b^{k-1})}{\sqrt{5}} + \frac{(a^k - b^k)}{\sqrt{5}}$$

$$= \frac{(a^{k-1} - b^{k-1})}{\sqrt{5}} + \frac{(a.a^{k-1} - b.b^{k-1})}{\sqrt{5}}$$

$$= \frac{((a+1)a^{k-1} - (b+1)b^{k-1})}{\sqrt{5}}$$

- We know that $(a+1) = a^2 = \frac{3+\sqrt{5}}{2}$ and $(b+1) = b^2 = \frac{3-\sqrt{5}}{2}$

  Substituting these values in the above equation we get

$$= \frac{((a^2)a^{k-1} - (b^2)b^{k-1})}{\sqrt{5}} = \frac{(a^{k+1} - b^{k+1})}{\sqrt{5}} = F_{k+1}$$

Hence proved by Induction that $\boldsymbol{F_n = \frac{(a^n - b^n)}{\sqrt{5}}}$

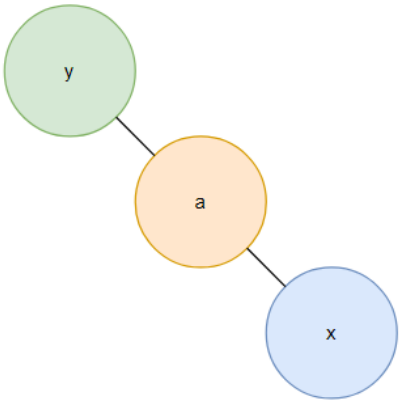
dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

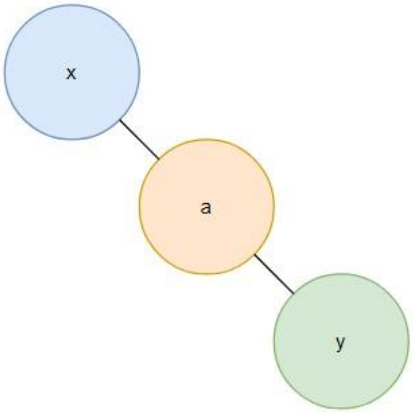
3. PROOF: by contradiction

- Given that T is BFS tree of a graph G and (x, y) is an edge of G where x & y are nodes in G
- Let us assume now that the level of x and y in T differs more than 1 i.e., |x-y|>1
- Following this we get the results diagrammatically given below as branches of the Tree T

Case 1: Branch of T1

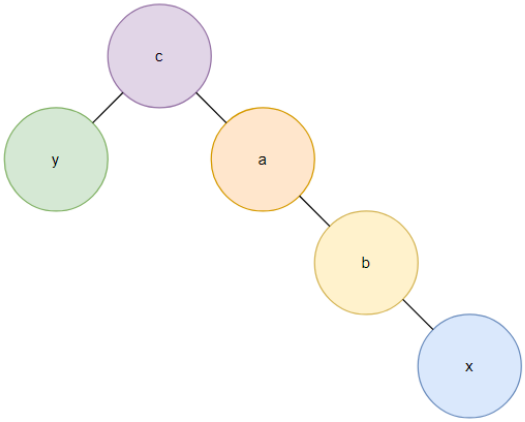


Case 2: Branch of T2



Case 3: Tree T3

- Assuming there is a difference between levels of x and y as 2 and x and y are connected with an edge
- We can have a Tree T3 as



    - From the tree we can say that since y was already traversed in the beginning and x was traversed later, they have a difference of levels as 2 but if there exists an edge between them this is simply not possible in BFS as in the tree BFS will visit x after visiting y and a. So, this tree is not possible
    - Hence proved by contradiction for Case 3
- As seen above in Case 1 and Case 2, x and y are not connected and so have no common edges between them
- By the definition of a BFS tree for a graph G**, this is a contradiction** which proves our initial assumption incorrect
- From all these cases we see that there is no possible way for the assumptions to be correct
- Hence, **we have proved by contradiction** that in Tree T the level of x and the level of y differ by at most 1.

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

4. PROOF: by contradiction

- **Given**: that in connected graph $G = (V, E)$, a specific vertex $u \in V$. After performing DFS and BFS respectively on graph G from vertex u we get the same Tree T.
- **Let us assume:** now that G has an edge (u, v) connecting nodes u and v which is not included in the Tree T
  - I. Now let us perform DFS on this same graph. We know that these nodes will appear on the same branch of the Tree T . They have an 'ancestral-descendant' node relationship.
  - II. Now let us perform BFS on this graph. We know that two nodes connected with an edge are always on the same level or one level apart from each other that is in the Tree T level of u and v differs by at most 1.
- Now combining the above two observations and the fact that we started the BFS and DFS from the root node u, we can say that u and v have a 'parent node-child node' relationship.
- But we assumed that (u, v) edge doesn't exist in Tree T
- **This is a contradiction** which proves our initial assumption incorrect
- Hence, we have proved by contradiction that G and T have the same structure and G cannot contain any edges that do not belong to T

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

5. **SOLUTION:**
   - **Given:** An unweighted, undirected and connected graph $G = (V, E)$.
   - **To find:** diameter of the graph and time complexity of the algorithm.
   - We can use Breadth-First Search for finding the shortest path between any two vertices in the given graph G
   - **Process:**
     - We can perform a loop to iteratively cycle through all the vertices and another for performing BFS through the given vertex and finding the maximum shortest distance from the chosen vertex to its adjacent vertices using a counter variable c. We have a visited array to keep a track of the vertices that have been visited.
     - We can set a variable maxDistance as 0 and update it whenever a value greater than it is found out
     - By the end of all iterations with all available vertices in the graph we will get our maximum shortest path's length i.e., the diameter of the graph
   - The Time Complexity in this algorithm would be $O(V * (V + E))$ because we visit every vertex (time complexity $O(V)$) and start performing BFS on it. And we know that the time complexity of BFS is $O(V + E)$
   - **Pseudocode:**

```
Set maxDistance = 0

for u in V

        Make a Queue for BFS
        Visited = [false for every v in {V}]
        Enqueue the queue with +(u)
        Visited[u] = true
        while Q≠[]
            c = 0
            v = Dequeue the queue
            for every w adjacent to v and visited[w] = False
                    c+=1
                    visited [w] = true
            end for

            maxDistance = max(maxDistance, c)

         end while

    end for

    return maxDistance
```

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

6. **SOLUTION:**

a. We have been given an undirected and unweighted graph $G = (V, E)$ with an edge
e ∈ E. We need to find an algorithm to determine whether the graph has a cycle. Since this graph is undirected, we can traverse it in any direction. Along with this we can name the nodes connected by the edge e as x & y . A cycle is found in a graph when in a path the first and the last vertex is the same.

Process:

- We first remove the edge e connecting x & y
- We now perform Breadth- First Search from node x in order to find node y
- If we found y after performing BFS from x, this would mean that there exists another path connecting x & y other than the edge e
- If we cannot reach y after performing BFS from x, we can safely say that there does not exist another path connecting them except e

Here, the time complexity for the algorithm is $O(V + E)$ as this is the time complexity of BFS algorithm and is the biggest operation being performed here. Along with this, we are running it once.

b. This algorithm wouldn't work if the graph is unweighted but directed. This is mainly because in a case where x & y are connected but directed from x to y ; if we apply BFS then the algorithm cannot backtrack and find the y vertex if we start from x vertex.

Modification:

- We first remove edge e connecting x & y
- We now perform BFS from vertex x and search for vertex y. If we reach vertex y and the edge is directed from y to x then we have found a cycle and we stop.
- If we do not reach y then we perform BFS from vertex y and search for x. If this works and the edge is directed from x to y then we stop. This works as there might be a cycle present but the direction might be opposite.
- If after performing BFS twice in the cycle, the BFS is not successful we can safely say that the edge e is not a part of any cycle

Here, the time complexity will still remain the same that is $O(V + E)$ as instead of performing BFS once we are performing it twice in the worst case.

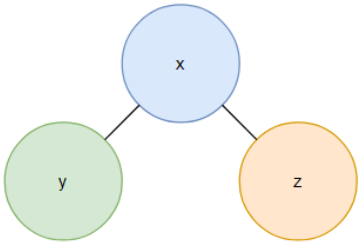
dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

7. **PROOF: by induction**

Base Case:

- Consider a binary tree with 3 nodes i.e., 1 root node and 2 leaf nodes
- Here number of nodes with 2 children is 1 which is 1 less than the number of leaves (2)

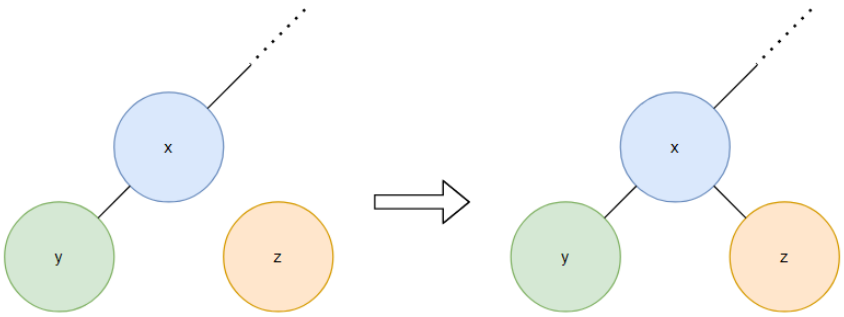


Induction Hypothesis:

- Assume that the given statement holds for a binary tree with k nodes in total

Induction Step:

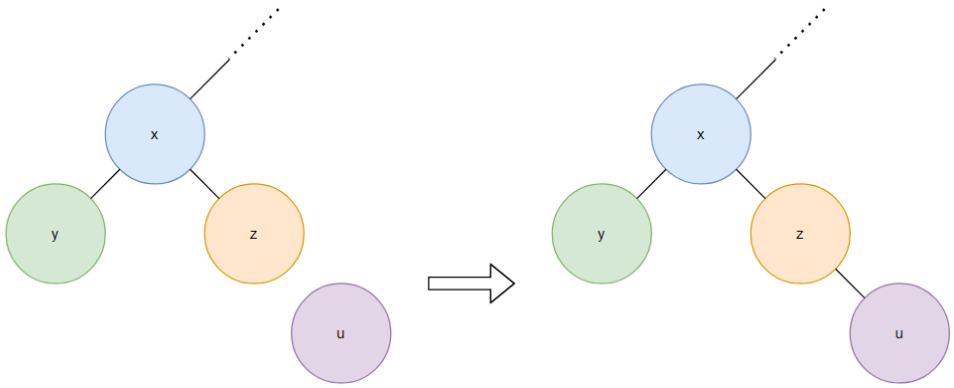
- We need to prove that the statement still holds true when we add a new node to the binary tree with k nodes
- Case 1: One of the parent nodes has only one child in the binary tree
  - Here adding $k+1^{th}$ node will lead to the leaf nodes increasing by 1 and parents with two children also increasing by 1. In the example given below, z is the $k+1^{th}$ node.



dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

- Case 2: All parents have two leaves in the binary tree
  - Here adding k+1th node will lead to creation of a new parent node with a single leaf node
  - This results in no change in the number of parent nodes with 2 children or the number of leaf nodes.
  - In the example given below, u is the k+1th node



- In both the cases the statement holds when we add k+1th node to the binary tree

Hence, proved by induction

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

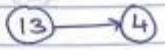
8. SOLUTION:

4.

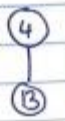
8 a. The sequence given is 4, 13, 7, 15, 21, 24, 10

Now, the first node to be inserted is 4 and it will be inserted as a single node binomial tree of 0 order.

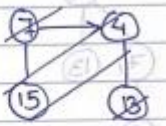
Insert 13 into heap

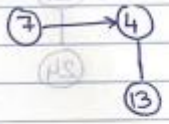


Here order matches so

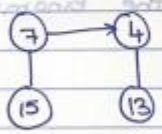


Insert 7 into heap





Insert 15 into heap, we get

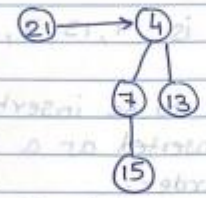


Here order matches, i.e. 1 so we alter the heap

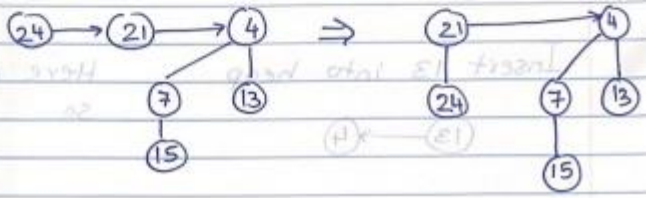
dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

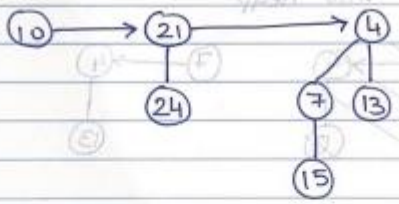
Insert 21, into heap



Insert 24, into heap



Insert 10, into heap



This is the binomial min-heap H1

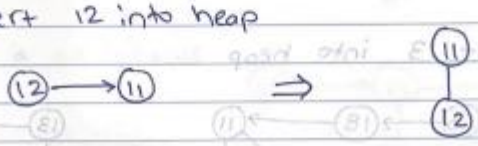
dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

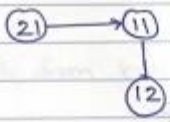
8 b The sequence given is 11, 12, 21, 24, 18, 13, 16

Insert 11 as a single node binomial tree of order 0

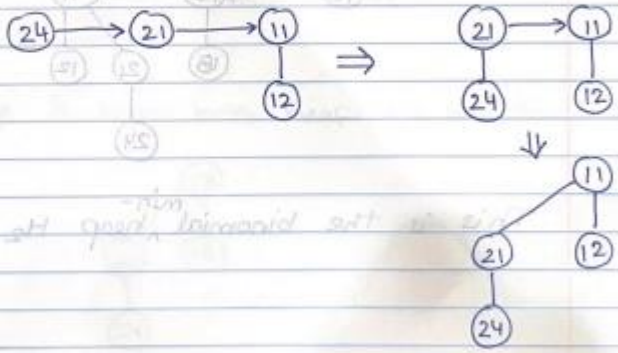
Insert 12 into heap



Insert 21 into heap



Insert 24 into heap

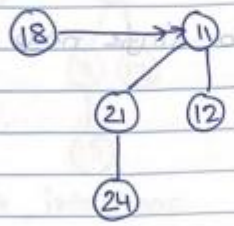


as they have same order

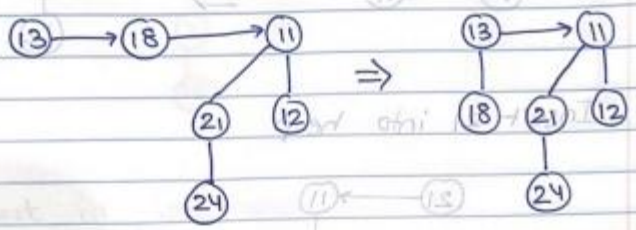
dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

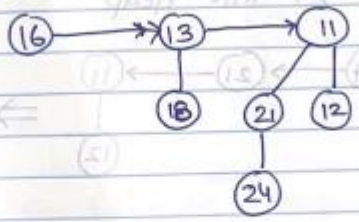
Insert 18 into heap



Insert 13 into heap



Insert 16 into heap



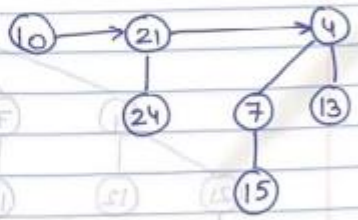
This is the binomial min-heap H_2

dnabar@usc.edu

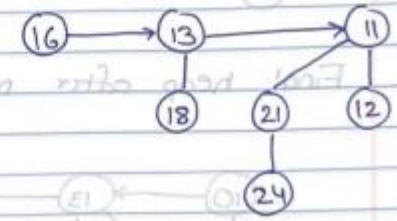
Deven Parag Nabar
USC ID: 7229446568

8.C. Merge H_1 and H_2

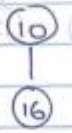
H_1 is as follows



H_2 is as follows

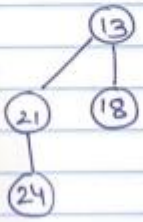


Merge B_0 from both heap H_1 and H_2



as $H_1 < H_2$

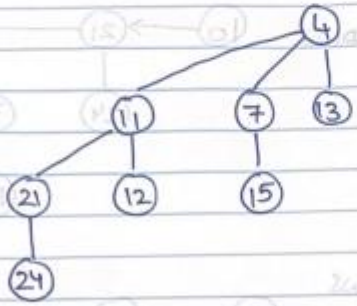
Merge B_1 from both heaps H_2 and H_2, $H_2 < H_1$



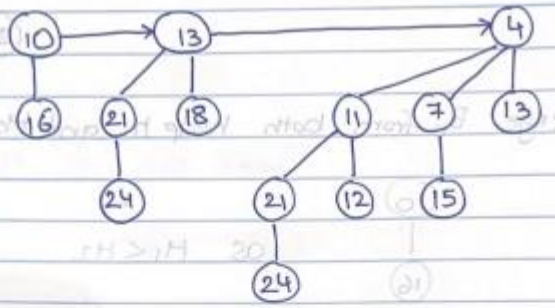
dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

Merge B_2 from H_1 and H_2, $H_1 \leq H_2$



Final heap after merging H_1 and H_2,



dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

9. SOLUTION:

Given: Considering an array on which following operations can be performed: addLast(e) which has a cost equal to 1 and deleteEveryThird() which removes third element in the list and costs equal to the number of elements in the array.

The worst sequence of operations is when you first add any number of elements and then continuously delete the third element by using the deleteEveryThird() operation till the array is empty

Using 2n tokens for addLast(e) and adding 8 elements we get

addLast(e)	**Tokens**	**Cost**	**Bank**
e = 1	2	1	1
e = 2	2	1	1
e = 3	2	1	1
e = 4	2	1	1
e = 5	2	1	1
e = 6	2	1	1
e = 7	2	1	1
e = 8	2	1	1
Adding 1,2,3,4,5,6,7,8	16	8	8

Now calculating the cost for deleting elements

deleteEveryThird()	**Cost**	**Bank (8)**
1, 4 and 7 deleted	8	8-8=0
2 and 6 deleted	5	0-5 = - 5
3 deleted	3	-5 -3 = -8
5 deleted	2	-8 – 2 = -10
8 deleted	1	-10 – 1 = -11

So, this doesn't work as the bank is going bankrupt

Using 3n tokens for addLast(e) and adding 8 elements we get

addLast(e)	**Tokens**	**Cost**	**Bank**
e = 1	3	1	2
e = 2	3	1	2
e = 3	3	1	2
e = 4	3	1	2
e = 5	3	1	2
e = 6	3	1	2
e = 7	3	1	2
e = 8	3	1	2
Adding 1,2,3,4,5,6,7,8	24	8	16

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

Now calculating the cost for deleting elements

deleteEveryThird()	**Cost**	**Bank (16)**
1, 4 and 7 deleted	8	16 – 8 =8
2 and 6 deleted	5	8 – 5 =3
3 deleted	3	3 - 3 = 0
5 deleted	2	0 – 2 = -2
8 deleted	1	-2 -1 = -3

So, this doesn't work either as the bank is going bankrupt again

Using 4n tokens for addLast(e) and adding 8 elements we get

addLast(e)	**Tokens**	**Cost**	**Bank**
e = 1	4	1	3
e = 2	4	1	3
e = 3	4	1	3
e = 4	4	1	3
e = 5	4	1	3
e = 6	4	1	3
e = 7	4	1	3
e = 8	4	1	3
Adding 1,2,3,4,5,6,7,8	32	8	24

Now calculating the cost for deleting elements

deleteEveryThird()	**Cost**	**Bank (24)**
1, 4 and 7 deleted	8	24 – 8 = 16
2 and 6 deleted	5	16 – 5 =11
3 deleted	3	11 - 3 = 8
5 deleted	2	8 – 2 = 6
8 deleted	1	6 -1 = 5

Since the balance in the bank is not negative, we can say that the token value chosen is correct (4n)

Sequence of operation is executing in constant time, amortized cost is $\boldsymbol{O}(\mathbf{1})$

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

10. PROOF:

- It is given that for a sequence of n operations the cost for the i^{th} operation is $i + j$ if $i = 2^j$ for $j \in N$ otherwise it is 1
  We can say that $j = \log i$ whenever the above condition is true
- The operations in a tabular form are as follows:

i^{th} operation	j	Total cost (i+j)
1	-	1
2	1	2+1 = 3
3	-	1
4	2	4+2 = 6
5	-	1
6	-	1
7	-	1
8	3	8+3 = 11
9	-	1
…	…	…

- We can see that there are 2 kinds of operations taking place over the sequence of n operations.
- Type 1 operations cost 1 each and the Type 2 operations cost $(i + \log i)$ and take place $\log n$ times. Hence Type 1 operations take place only $(n - \log n)$ times
- The amortized cost of an operation is given by $\frac{T(n)}{n}$. Let us find $T(n)$
- $T(n) = 1.(n - \log n) + (2^1 + 2^2 + 2^3 + 2^{\log n}) + (\log 2 + \log 4 + \log 8 + \log 16 + \log n)$
- So, $T(n)$
  $= n - \log n + \sum_{i=1}^{\log n} 2^i + \sum_{i=1}^{\log n} \log 2^i \leq n - \log n + (2^{\log n+1} - 2)$ + n
  $\leq n - \log n + 2n - 2 + n$
  $\leq 4n$ -2 - $\log n$
- As $n \to \infty$, except 4n, all other terms grow slower in the equation and can be ignored.
- **Amortized Cost =** $\frac{T(n)}{n} = \lim_{n\to\infty} \frac{4n - 2 - \log n}{n} = \frac{4n}{n} = 4 = \boldsymbol{O(1)}$ **which is constant time.**
- Hence the given statement is proved.