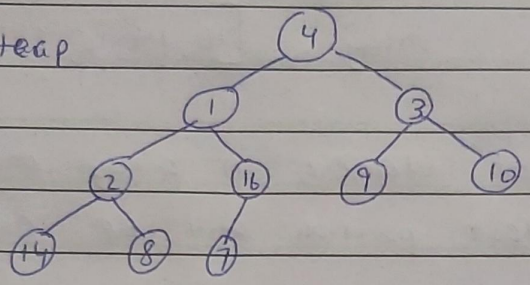
Date:
M T W T F S S

# DAA ASSIGNMENT 2

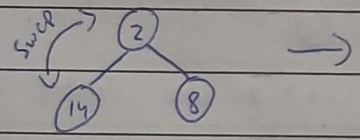
Muhammad Safdar 22K-4304 BCS-5B

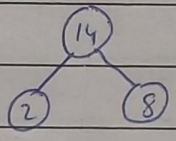
Q1) A = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]

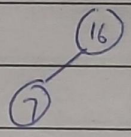
Corresponding Heap




Making Max Heap (Parent > child)
Starting from leaf nodes

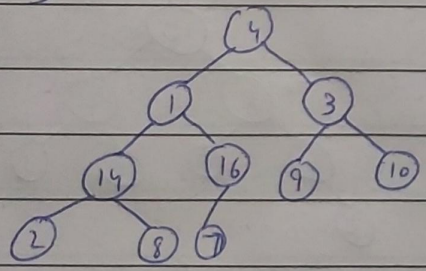


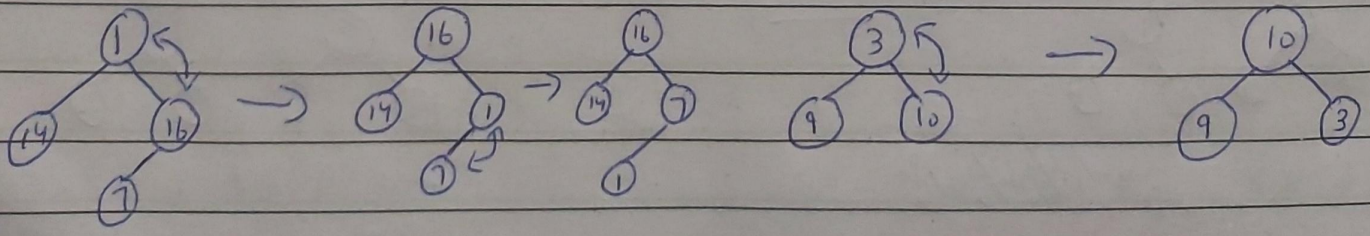




Already
Parent > child.

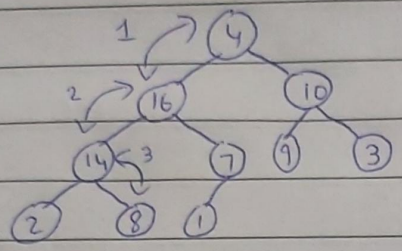
Resulting Heap

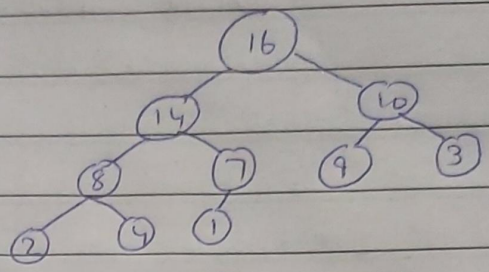



Now we will move to upper level



Date:
M T W T F S S

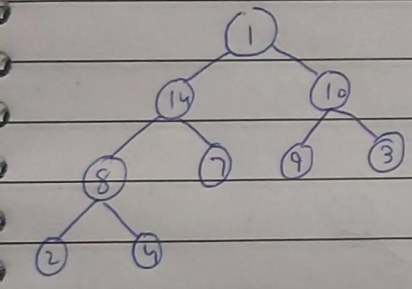


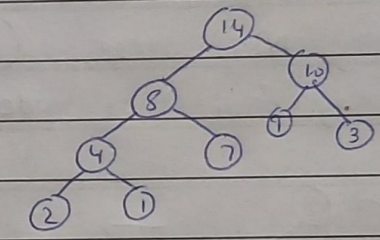
MAX HEAP

Array [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]

Now Swap 1st and last element in array and delete last from heap. Array at last position will be considered sorted.

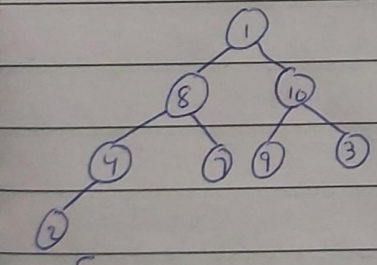


Heapify →

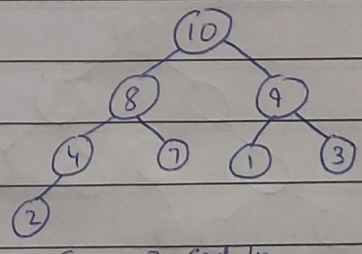


Swap 14 & 1 and delete 14

[1, 14, 10, 8, 7, 9, 3, 2, 4, ~~16~~]

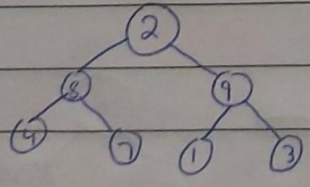


Heapify →

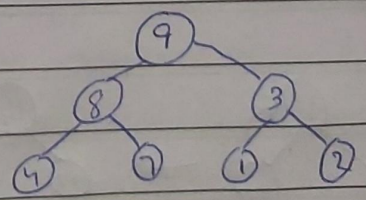


Swap 2 and 10
Delete 10

[1, 8, 10, 4, 7, 9, 3, 2, ~~14~~, ~~16~~]



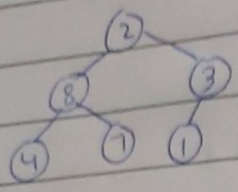
Heapify →



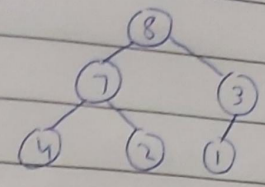
Swap 9 and 2
Delete 9

[2, 8, 9, 4, 7, 1, 3, ~~10~~, ~~14~~, ~~16~~]

Date:
M T W T F S S



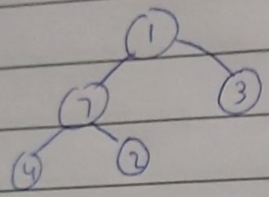
Heapify →



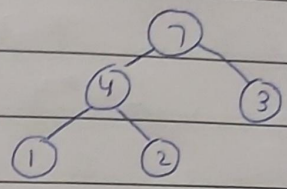
Swap 1 & 8

Delete 8

[2, 8, 3, 4, 7, 1, 9, 10, 14, 16]



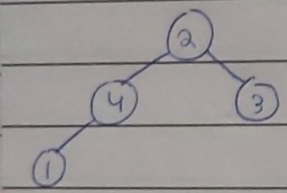
Heapify →



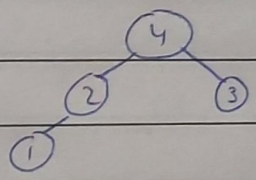
Swap 7 & 2

Delete 7

[1, 7, 3, 4, 2, 8, 9, 10, 14, 16]



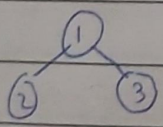
→



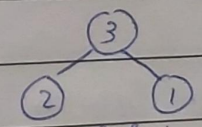
Swap 4 & 1

Delete 4

[2, 4, 3, 1, 7, 8, 9, 10, 14, 16]



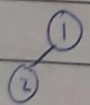
→



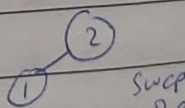
Swap 3 & 1

Delete 3

[1, 2, 3, 4, 7, 8, 9, 10, 14, 16]



→



Swap 1 & 2

Delete 2

[1, 2, 3, 4, 7, 8, 9, 10, 14, 16]

(1)

Delete 1 →

[1, 2, 3, 4, 7, 8, 9, 10, 14, 16]

Sorted Array.

Date:
M T W T F S

Q2a A = [1, 2, 3, 4, 5, 6, 7, 8]

Binary Search : Find x = 8

L = 0 mid = 3 R = 7

iteration 1: x = = A[3] NO

x > A[3] YES

L = 4 mid = 5 R = 7

iteration 2: x = = A[5] NO

x > A[5] YES

L = 6 mid = 6 R = 7

iteration 3 : x = = A[6] NO

x > A[6] yes

L = 7 mid = 7 R = 7

iteration 4 : x = = A[7] YES

Jump Search Find x = 8

Jump size K = 4

~~while (arr[min]~~

x <= 0 NO

x <= 4 NO

x <= n = 7 YES

Now applying linear search from 5 to 7

Date:
M T W T F S S

$x == A[5]$ NO
$x == A[6]$ NO
$x == A[7]$ YES Answer

| | Time Complexity | Space Complexity |
|---|---|---|
| Binary Search | $O(\log n)$ | $O(1)$ |
| Jump Search | $O(\frac{n}{k} + k)$ | $O(1)$ |

Both the algorithms are more efficient way of searching compared to linear search and requires no additional space. but for both the techniques array needed to be sorted.

Binary Search can outperform Jump search when array size is large or when the target is evenly distributed. It also works better when target is near end.

Jump Search can outperform binary search when array size is small or when target is located near start of an array.

Date:
M T W T F S S

Q2b Interpolation Search:-

1. It is the improved technique of Binary search where instead of always dividing into two halves it estimates the position of target element by using lowest and highest values. It uses formula:-

$$mid = L + (x - Arr[L] \times (H-L) / (Arr[H] - Arr[L])$$

$x$ = target element  $L$ = Low  $H$ = high

2 It is useful when searching through uniformly distributed data such as records in the phone book where names are alphabetically ordered

3 Time Complexity

Best Case: $O(1)$

Average Case: $O(\log(\log(n)))$

Worst Case: $O(n)$

It is highly dependend on the uniform data distribution. when data is not evenly spread it performs poorly.

Exponential Search:-

1- It is the combination of jump and binary search. it begins by searching for the range where element lies by jumping by powers of 2 (1, 2, 4, 8, 16...) Once the range is found it performs binary search on it.

Date:
M T W T F S S

2- This is ideal for searching in large dataset where target is near the begining such as ~~if you are~~ looking for specific log entry in time ordered log file.

3- Time Complexity

Best Case: O(1)

Average Case: O(log n)

worst Case: O(log n)

This search does not rely on uniform data distribution So it does not effects it's performance.

Q3 ID : 22K-4304

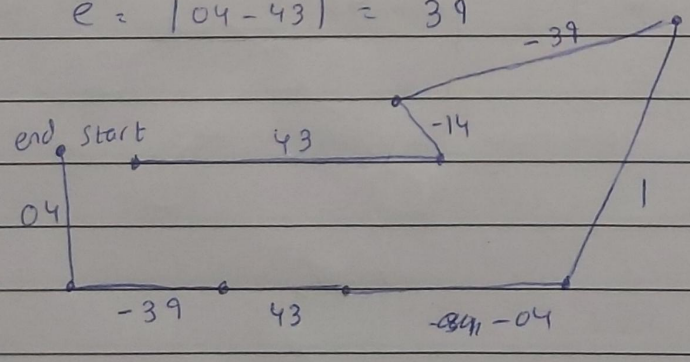
a = 43

b = 04

c = 43 x 7 mod 60 = 01

d = 4304 mod 30 = 14

e = |04 - 43| = 39



Date:
M T W T F S S

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 43 | -14 | -39 | 1 | -4 | 43 | -39 | 4 |
| 1 | | | | 1 | -4 | | | -4 |
| -38 | | | -39 | 1 | -4 | 43 | | 39 |
| -52 | | -14 | -39 | 1 | -4 | 43 | -39 | 0 |
| -9 | 43 | -14 | -39 | 1 | -4 | 43 | -39 | 4 | 4 |

CSS = 40

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 43 | -14 | -39 | 1 | | -4 | 43 | -39 | 4 |
| -14 | | -14 | -39 | | -39 | 43 | 43 | -39 | -39 |
| 29 | 43 | -14 | -39 | 1 | -38 | 39 | -4 | 43 | -39 | 4 | -35 |

CSS = 29

CSS = 43

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 43 | -14 | -39 | 1 | -4 | 43 | -39 | 4 |

CSS = 43 | CSS = 1 | CSS = 43 | CSS = 4

Highest CSS = 43 so maximum Sub Array = 43

Date:
M T W T F S S

Q4 C = [43, 4304, 04, 30, 304, 22]

= [dc, dcd, d, c, cd, bb]

we can apply radix sort.

Step 1: make all elements 3 letters by adding blanks at start.

C = [_dc, dcd, __d, __c, _cd, _bb]

Step 2: Sort by last alphabet.

C = [_bb, _dc, __c, dcd, __d, _cd]

Step 3: Sort by 2nd Alphabet.

C = [__c, __d, _bb, dcd, _cd, _dc]

Step 4: Sort by 1st Alphabet

C = [__c, __d, _bb, _cd, _dc, dcd]

Find Sorted Array

C = [c, d, bb, cd, dc, dcd]

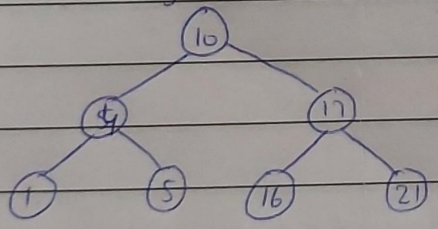
Date:
M T W T F S S

Q5 Arr = [1, 0, 4, 0, 0, 0, 2, 0]

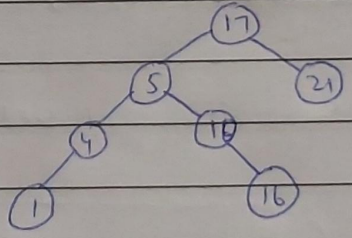
```
nonZeroIndex = 0;

for (i=0; i < arr.length(); i++) {

    if (Arr[i] != 0) {
        Arr[nonZeroIndex] = Arr[i];
        nonZeroIndex ++;
    } }

while (nonZeroIndex < Arr.length()) {
    Arr[nonZeroIndex] = 0;
    nonZeroIndex ++;
}
```

Q6

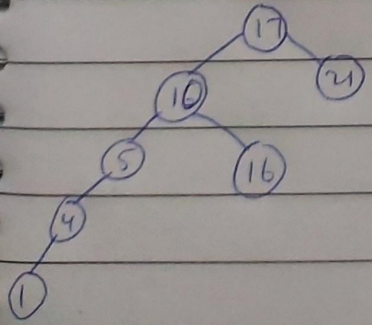
Height = 2



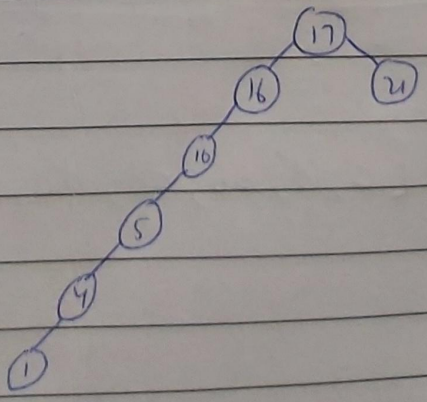

Height = 3




Height = 4




Height = 5



Date:
M T W T F S S

Q.7 Assuming Array range is from 0 to N-1

```
Arr[100] = {5, 6, 10, 14 .....}

frequency[N] = {0};

for (int i=0; i<100; i++) {
    if (freq[Arr[i]] == 1) {
        return Arr[i]      // Duplicate element.
    }
    else {
        frequency[Arr[i]] += 1;
    }
}
return 0;      // No duplicates.
```