# ALGO ASSIGNMENT #02.

BASIL ALI KHAN
20K-0477

Question # 01

⇒ Bruteforce (O(n²)):

```
int func (int arr[], int n) {
    int profit=0, temp;
    for (i=0 to n-1) {
        temp = 0;
        for (j=0 to n-1) {
            temp = temp + arr[j];
            if (temp > profit) {
                sum = temp;
            }
        }
    return profit;
}
```

⇒ Dynamic (O(n)):

```
int func (int arr[], int n) {
    int temp[n];
    temp[0] = arr[0];
    for (i=0 to n-1) {
        if (arr[i] + temp[i-1] > 0) {
            temp[i] = arr[i] + temp[i-1].
        }
        else {
            temp[i] = arr[i]
        }
        for (i=0 to n-1) {
            profit = max (profit, temp[i]);
        }
    return profit; }
```

Question #02

⇒ Algorithm O(nlogn):

```
void func (int arr[], int n) {
    MergeSort (arr);          → nlogn
    for (i=0 to n-1) {
        int temp = arr_unsorted arr[i];
        if (Binary search (arr, 0, n-1, temp))
            print result.
}.
```

⇒ Algorithm O(n)

→ Push one array in mordered set <int> temp.

```
for (i=0 to arr.len() -1) {
    if (temp.( value_arr2 [i] ! temp=end.u )
        print
}
```

Question #03

⇒ Binery search

```
int Search (int arr[], int l, int h) {
    if (h >= l) {
        mid = (h+l) /2;
        if (mid == arr [mid] {
            return mid;
        }
        if (mid +1 <= arr[h]) {
            result = Search (arr, mid+1, h);
        }.
```

```
if (mid-1 >= arr[l]{
    return search(arr, l, mid-1);
}
return -1;
}
```

Question #04:

```
int func(int arr, int n){
    int index;
    for( ~~i=0~~ i=0 to n-1){
        if (arr[i] != 0){
            swap(arr[index], arr[i])
            index++;
        }
    }
```

Question # 05

⇒ Binary Search:

low = 0, high = 9, mid = 4

A[mid] ≠ 9 → low = 5 as b/c A[mid] < 9.

low = 5, high = 9, mid = 7.

A[mid] ≠ 9 → low = 8 b/c A[mid] < 9.

low = 8, high = 9, mid = 8

A[mid] ≠ 9 → low = 9 b/c A[8] < 9.

low = 9, high = 9, mid = 9

A[9] = 9 ⇒ found.

⇒ Jump Search :

element to be searched => 100

prev = 0 , step size = 2 → $\sqrt{size} = 2.8 = 2$

⇒ 1, 4, 5, 6, 9, 10, 11, 100 :

Arr[1] < 100 .

prev = 2 step_size = 4.

⇒ 1, 4, 5, 6, 9, 10, 11, 100

arr[3] < 100

prev = 4 , step_size = 6.

⇒ 1, 4, 5, 6, 9, 10, 11, 100

arr[6] < 100

prev 6 , step_size = 8.

Now liner search from prev to step_size.

⇒ 1, 4, 5, 6, 9, 10, 11, 100

⇒ arr[prev] == 100

⇒ return arr[prev]

→ Jump Search :

→ less comparison than liner search

→ only works on sorted array

→ Worst case complexity $O(\sqrt{n})$

→ If element in beginning than Jump search will be faster.

→ Binary Search :

→ less comparisons than liner search

→ needs stacks space for recursive method.

→ If element in middle than binary search is faster.

INTERPOLATION SEARCH:

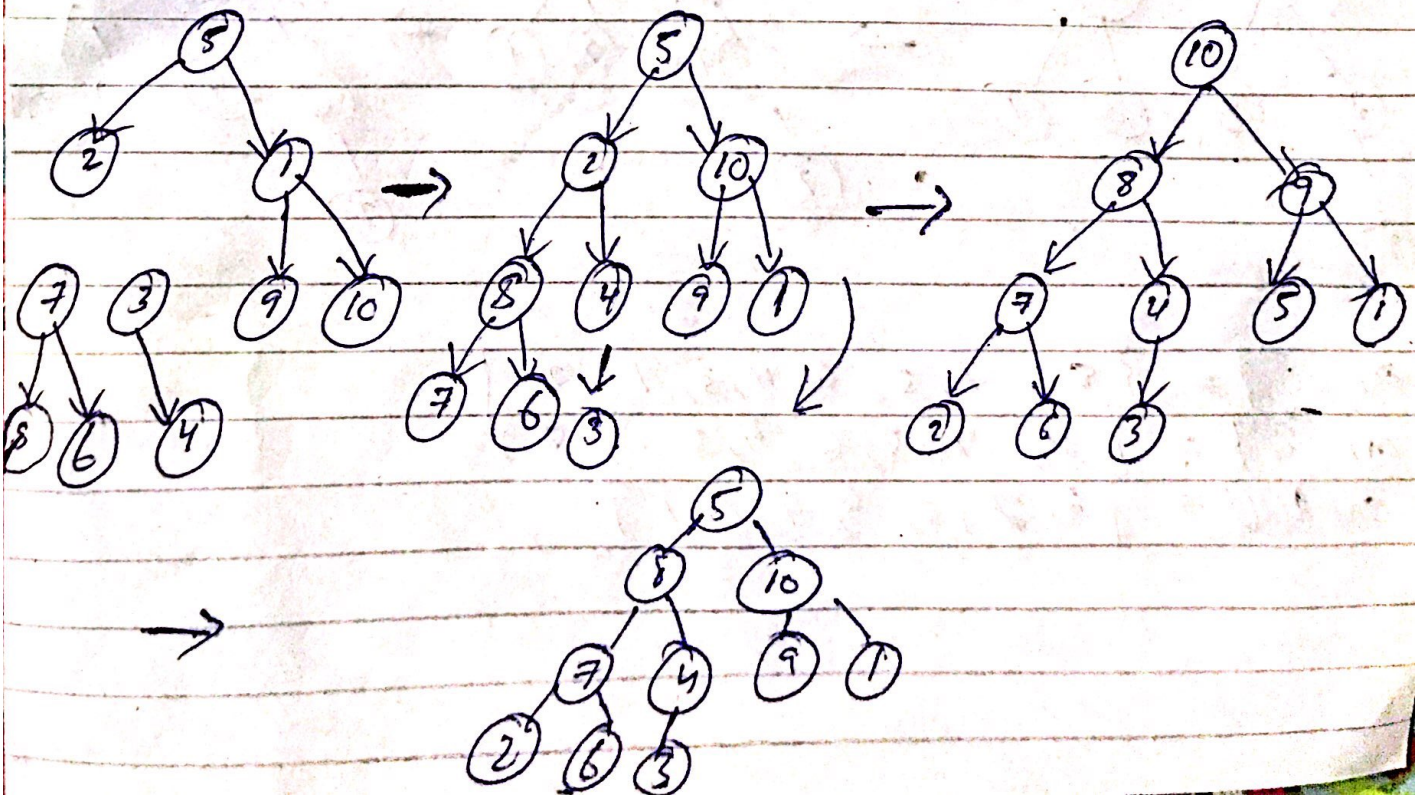
→ Used when sorted values are evenly distributed.
→ Creates new pivot values to reduce number of searches. Select closest pivot to the value.
→ time complexity → O(log(logn)) → average case.
→ O(n) → worst case

Exponential Search:

→ Find the range where element may lie
→ then do binary search
→ Starts array with 1 element then double it
→ Keep on doubling until last element greater than value to find.
→ then binary search

Question #06:

arr[10] = {5, 2, 1, 7, 3, 9, 10, 8, 6, 4}.



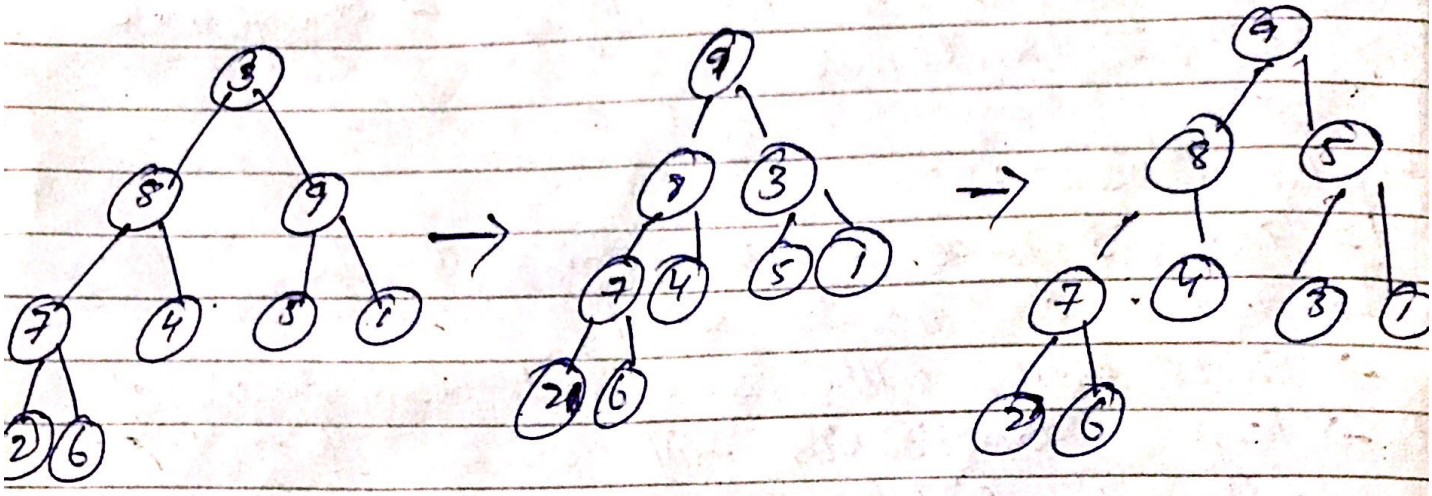
arr[10] = {10, 8, 9, 7, 4, 5, 1, 2, 6, 3}

→ swap first & last element

arr[10] = {3, 8, 9, 7, 4, 5, 1, 2, 6, 10}

→ then remove last when making heap

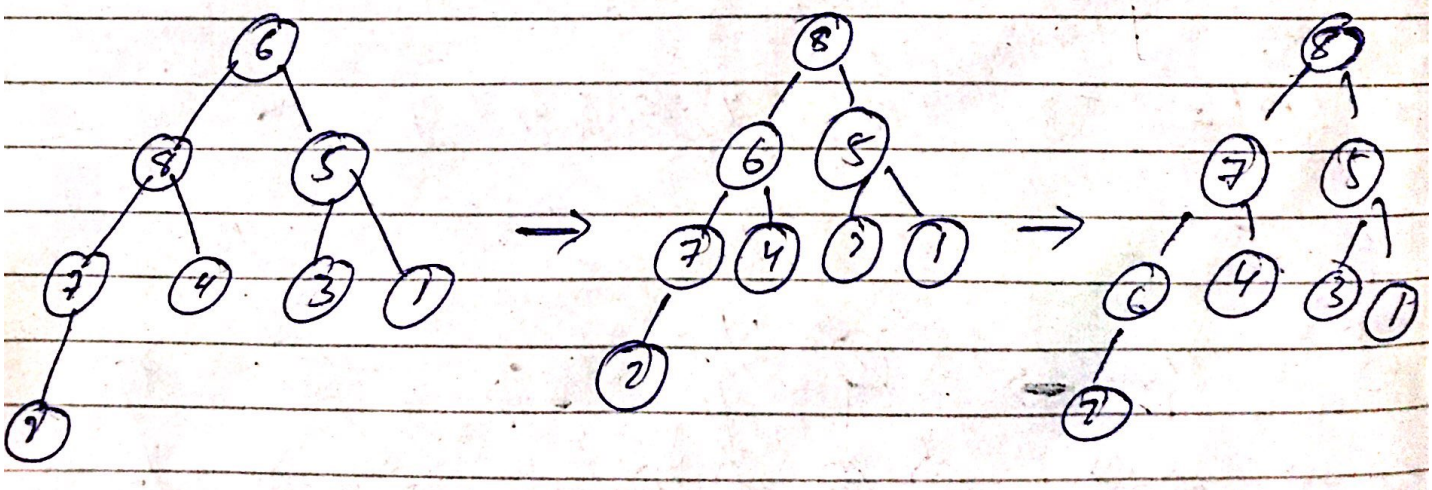
arr[10] = {8, 8, 9, 7, 4, 5, 1, 2, 6, 10}



arr = {9, 8, 5, 7, 4, 3, 1, 2, 6}

arr = {6, 8, 5, 7, 4, 3, 1, 2, 9}

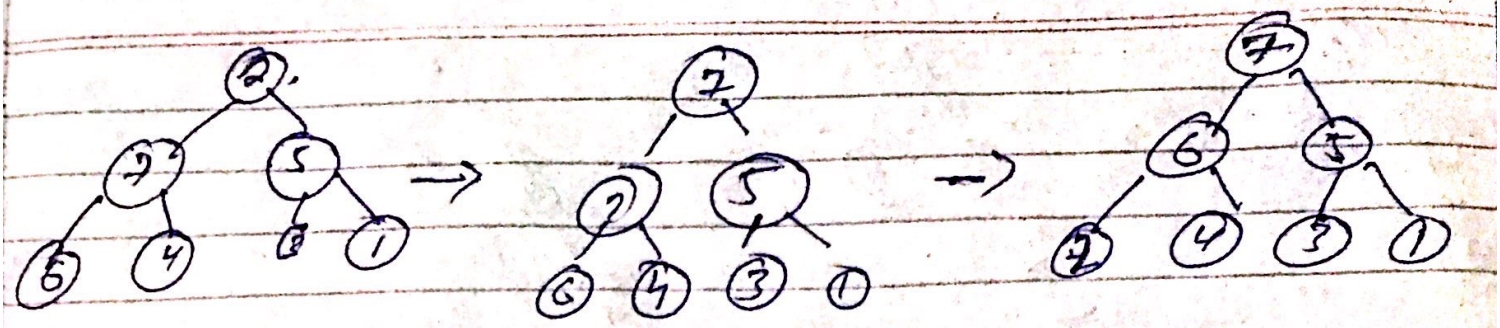
arr = {6, 8, 5, 7, 4, 3, 1, 2}



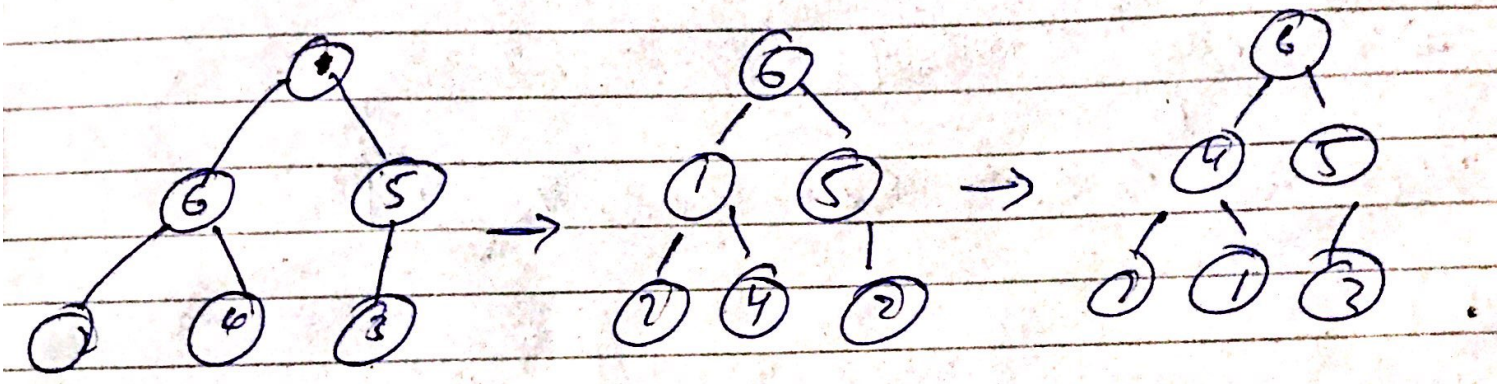
arr = {8, 7, 5, 6, 4, 3, 1, 2}

arr = {2, 7, 5, 6, 4, 3, 1, 8}

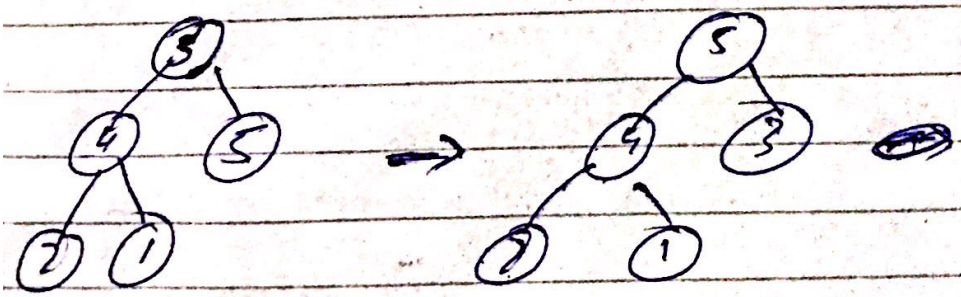
arr = {2, 7, 5, 6, 4, 5, 1}



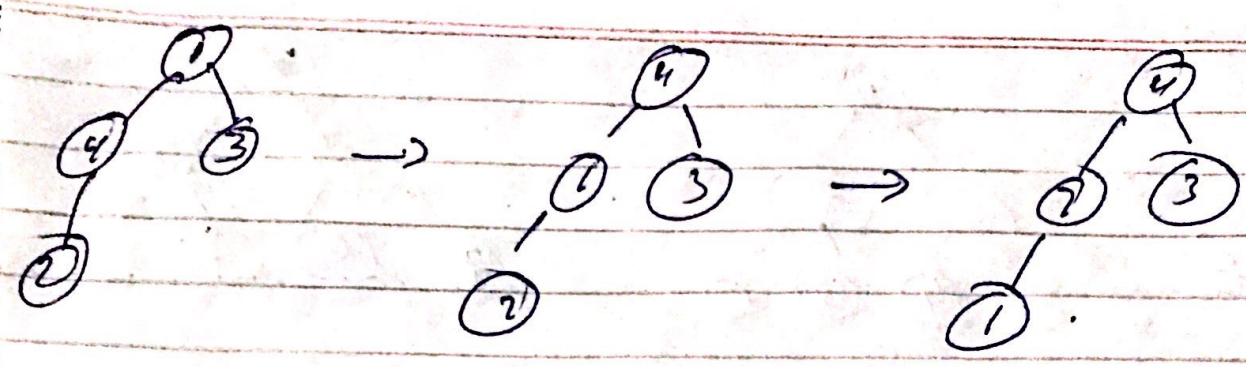
arr [redacted] = {7, 6, 5, 2, 4, 5, 1}
arr [redacted] = {1, 6, 5, 2, 4, 3, 7}
arr [redacted] = {1, 6, 5, 2, 4, 3}



arr [redacted] = {6, 4, 5, 2, 1, 3}
arr [redacted] = {3, 4, 5, 2, 1, 6}
arr = {3, 4, 5, 2, 1}



arr = {5, 4, 3, 2, 1}
arr = {1, 4, 5, 2, 5}.
arr = {1, 4, 3, 2}



arr = { 4, 2, 3, 1 }

arr = { 1, 2, 3, 4 }

~~arr =~~ arr = { 1, 2, 3 }

⇒ Array Sorted

arr[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }