Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2

Q1 $O(N^2)$

```
#include <iostream>
using namespace std;


int checkAllNegative(int arr[], int n,int &smallestNeg)
{
     bool allNegative = true;
     //check if there is a positive number
     for(int i=0;i<n;i++)
     {
          if(arr[i] > 0)
          {
               allNegative = false;
               break;
          }

          if(arr[i]<smallestNeg)
               smallestNeg = arr[i];
     }
     return allNegative;
}

int main()
{
    int arr[] = {1,-2,-3,-4,1,1,-6,-8,1,-8,-2,-3};
    int n = sizeof(arr)/sizeof(int);

    int smallestNeg = 0;

    if(checkAllNegative(arr,n,smallestNeg))
          {
               cout<<smallestNeg;
               exit(1);
          }

    int currMax = 0;
    int prevMax = 0;

    for(int i=0;i<n;i++)
    {
        for(int j=i;j<n;j++)
```

Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2

```
        {
            currMax+=arr[j];
            if(currMax>prevMax)
                prevMax = currMax;
        }

        currMax = 0;
    }

    cout<<prevMax;


    return 0;
}
```

Q1 O(N)

```
#include <iostream>
using namespace std;

int checkAllNegative(int arr[], int n,int &smallestNeg)
{
     bool allNegative = true;
     //check if there is a positive number
     for(int i=0;i<n;i++)
     {
          if(arr[i] > 0)
          {
               allNegative = false;
               break;
          }

          if(arr[i]<smallestNeg)
               smallestNeg = arr[i];
     }
     return allNegative;
}

int main()
{
```

# Assignment 2

```
    int posSum=0;
    int negSum=0;

    int arr[] = {1,-2,-3,-4,1,1,-6,-8,1,-8,-2,-3};
    int n = sizeof(arr)/sizeof(int);

     int smallestNeg=0;

     if(checkAllNegative(arr,n,smallestNeg))
          {
               cout<<smallestNeg;
               exit(1);
          }

     int prevSum=0;
     int currSum=0;

     for(int i=0;i<n;i++)
     {
          currSum+=arr[i];
          if(currSum > prevSum)
               prevSum = currSum;

          if(currSum < 0)
               currSum = 0;
     }
     cout<<prevSum;
    return 0;
}
```

Q2 O(NLog(N))

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;

void MergeSortedIntervals(vector<int>& v, int s, int m, int e) {

    // temp is used to temporary store the vector obtained by merging
    // elements from [s to m] and [m+1 to e] in v
    vector<int> temp;
```

Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2

```
    int i, j;
    i = s;
    j = m + 1;

    while (i <= m && j <= e) {

        if (v[i] <= v[j]) {
            temp.push_back(v[i]);
            ++i;
        }
        else {
            temp.push_back(v[j]);
            ++j;
        }

    }

    while (i <= m) {
        temp.push_back(v[i]);
        ++i;
    }

    while (j <= e) {
        temp.push_back(v[j]);
        ++j;
    }

    for (int i = s; i <= e; ++i)
        v[i] = temp[i - s];

}

void MergeSort(vector<int>& v, int s, int e) {
    if (s < e) {
        int m = (s + e) / 2;
        MergeSort(v, s, m);
        MergeSort(v, m + 1, e);
        MergeSortedIntervals(v, s, m, e);
    }
}

void SortArrays(vector<int>& A, vector<int>& B)
```

# Assignment 2

```
{
    MergeSort(A,0,A.size()-1);
    MergeSort(B,0,B.size()-1);
}

void SeparateDigits(int num,vector<int>&arr)
{
    while(num>0)
    {
        arr.insert(arr.begin(),num%10);
        num/=10;
    }
}

bool sumOfArraysEqualToTarget(vector<int> A,vector<int> B, int sum)
{
    int n = A.size();
    int m = B.size();
    int l=0;
    int r=m-1;

    while (l < n && r>=0) {
        if (A[l] + B[r] == sum)
        {
          cout <<A[l]<<" in A"<<" and "<<B[r]<<" in B";
          return 1;
        }
        else if (A[l] + A[r] < sum)
            l++;
        else // A[i] + A[j] > sum
            r--;
    }

    cout<<"Not Possible";
    return 0;
}


int main()
{
    int a = 3124;
    int b = 5162;
```

# Assignment 2

```
    vector<int> A;
    vector<int> B;

    SeparateDigits(a,A);
    SeparateDigits(b,B);

    //Sort Arrays
    SortArrays(A,B);

    sumOfArraysEqualToTarget(A,B, 10);

    return 0;
}
```

Q2 O(N)

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

void SeparateDigits(int num,vector<int>&arr)
{
    while(num>0)
    {
        arr.insert(arr.begin(),num%10);
        num/=10;
    }
}

bool sumOfArraysEqualToTarget(vector<int> A,vector<int> B, int sum)
{
    int i=0;
     unordered_map <int,int> hash;

    for(int i=0;i<A.size();i++)
    {
        hash[A[i]] = i;
    }
```

Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2

```
    for(int i=0;i<B.size();i++)
    {
        int numToFind = sum-B[i];
        if(hash.find(numToFind)!=hash.end())
        {
            cout <<hash[numToFind]<<" index in A"<<" and "<<i<<" index in B";
            return 1;
        }
    }

    cout<<"NOT FOUND";
    return -1;
}


int main()
{
    int a = 3124;
    int b = 5162;

    vector<int> A;
    vector<int> B;

    SeparateDigits(a,A);
    SeparateDigits(b,B);

    sumOfArraysEqualToTarget(A,B, 10);

    return 0;
}
```

Q3 O(LogN)

```
#include <vector>
#include <iostream>
using namespace std;
int findLeftMost(vector<int> a)
{
    int n = a.size();
    int leftmost = -9999999;
    int l = 0;
```

Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2

```
    int r = n - 1;
    int m;
    while (l <= r)
    {
        m = (l + r) / 2;
        if (a[m] == m)
        {
            leftmost = m;
            r = m-1;
            //rightmost = m;
            //l = m+1;
        }
        else if (a[m] >m)
        {
            r= m - 1;
        }
        else
        {
            l = m + 1;
        }
    }
    return leftmost;
}

int main()
{

    vector<int> a = {-1,-2,-3,3,4,5};
    int b = findLeftMost(a);
    cout << b;
}
```

Q4 O(N) time and O(1) Space

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> func(vector<int> A)
{
    int n = A.size();
    int k = 0;
    int count = 0;
```

# Assignment 2

```
    for (int i = 0; i < n; i++)
    {
        if (A[i] > 0)
        {
            A[k] = A[i];
            k++;
        }
    }

    for (k + 1; k < n; k++)
    {
        A[k] = 0;
    }
    return A;
}

int main()
{
    vector<int> a = {0, 1, 0, 2, 3, 0, 7, 10, 0, 1};
    int n = a.size();
    for (int i = 0; i < n; i++)
    {
        cout << ' ' << a[i] << ' ';
    }
    cout << endl;
    a = func(a);

    for (int i = 0; i < n; i++)
    {
        cout << ' ' << a[i] << ' ';
    }
}
```

Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2

Q5

Q5 $\boxed{x=30}$

Jump Search

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 | 12 | 20 | 30 |

$n = 8$

$\sqrt{n} = 2$ ← Jump size

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 | 12 | 20 | 30 |

not found becase arr[i] ≤ x

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 | 12 | 20 | 30 |

Not found becuse arr[i] ≤ x

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 | 12 | 20 | 30 |

Not found because arr[i] ≤ x

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 | 12 | 20 | 30 |

Can't jump furter so apply linear search in that block to find it of O(N).

Binary Search

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 | 12 | 20 | 30 |

arr[i] ≤ 30

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| 9 | 12 | 20 | 30 |

entire left sub array is disregarded.

arr[i] ≤ 30

$\boxed{30}$

∴ found

arr[i] = x

Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2

Pros & Cons:-

- Both need to be sorted (arr)
- JumpSearch best time complexity $\sqrt{n}$, & worst is between $O(N)$ to $O(\log n)$
- Jump search is better when you only traverse back once i.e $O(N)$ times but binary search traverses $O(\log n)$ times. i.e in the case of smallest element.
- Otherwise binary search works better than Jump search.

Interpolation Search

- Works on sorted Array
- ~~B~~ Better for uniform array
- Uses this formula to search $pos = low + \frac{(key - arr[i]) * (\cancel{low}\ high - low)}{(arr[high] - arr[low])}$
- Uses a range between low to high to find the element.
- best case $O(1)$, Avg $O(\log(\log n))$, word ( $O(N)$ )
- It returns arr[high] when closer to high and arr[low] when closer to arr[low].
- if arr[pos] > x, low = pos+1, else high = pos-1

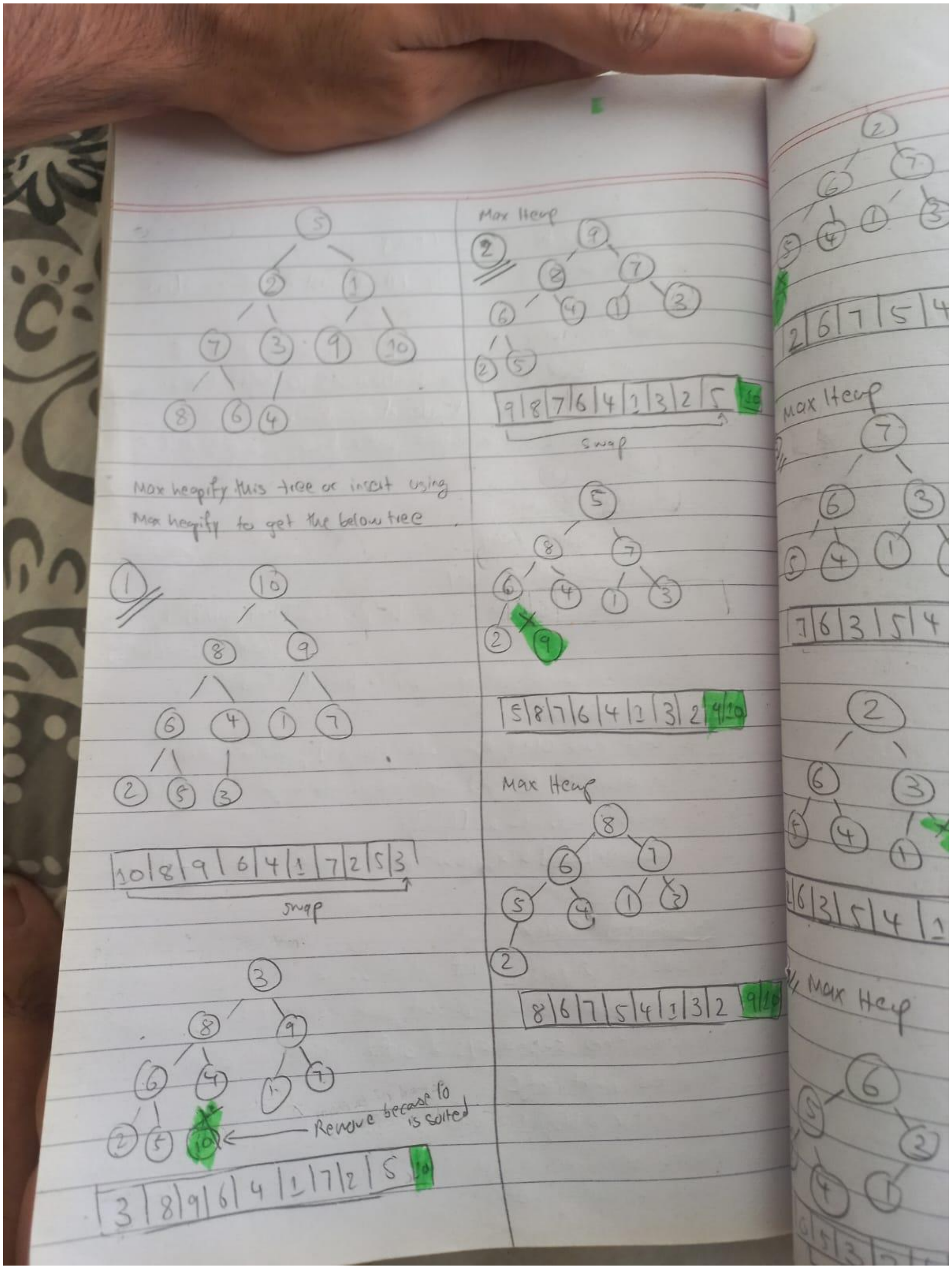
Exponential Search

- works in 2 parts, first it finds 2 in rage in which element to be found is present then we apply binary search
- Trying to find sub-arrays of different sizes in order to find the right range
- works for $O(\log n)$ and for unbounded/infinite sizes
- works better than binary search especially when element is closer to starting index.

Mohsin Ali Mirza K200353 5E-BSCS

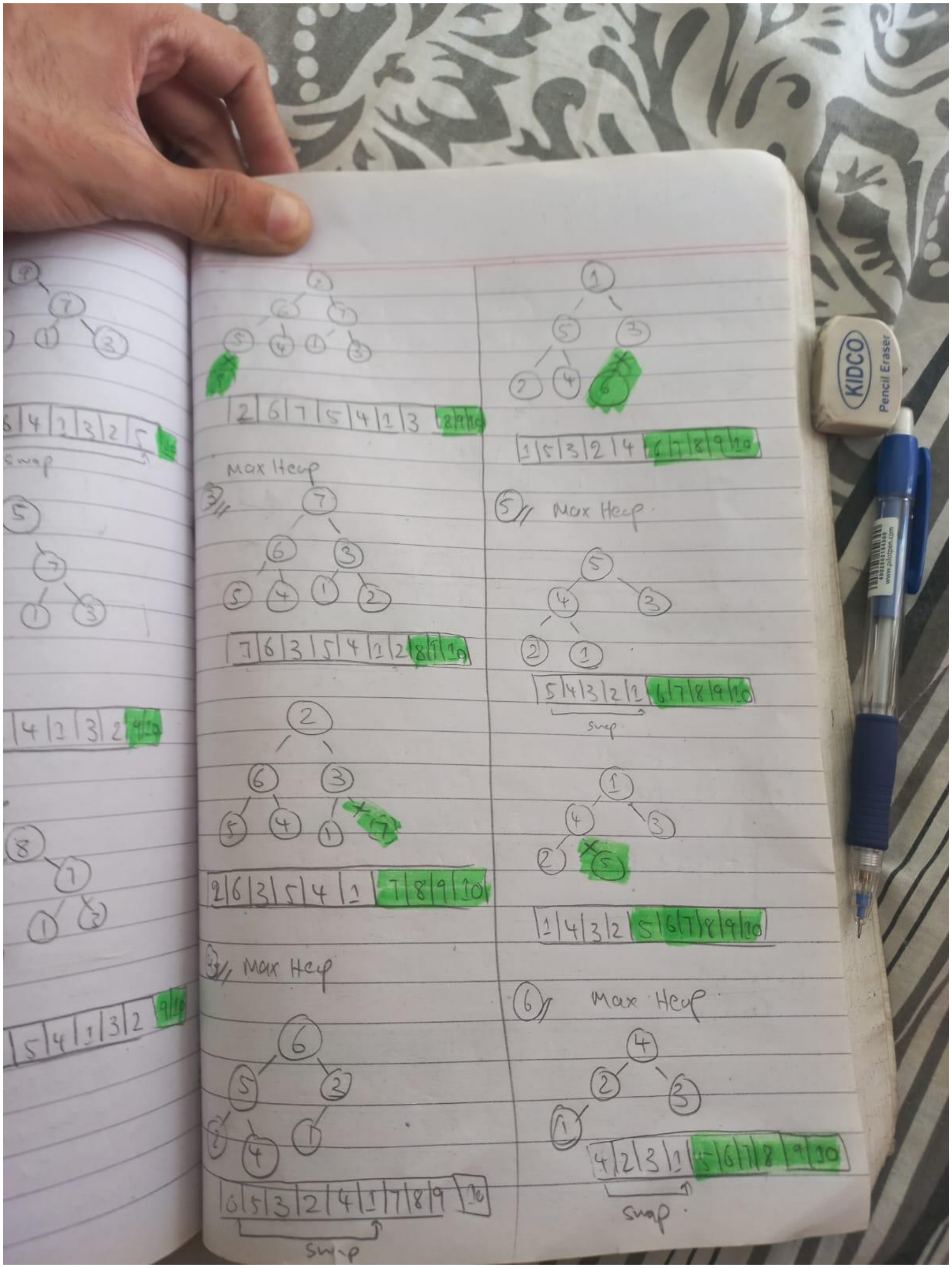
# Assignment 2

Q6



Mohsin Ali Mirza K200353 5E-BSCS

# Assignment 2



Mohsin Ali Mirza K200353 5E-BSCS

# **Assignment 2**