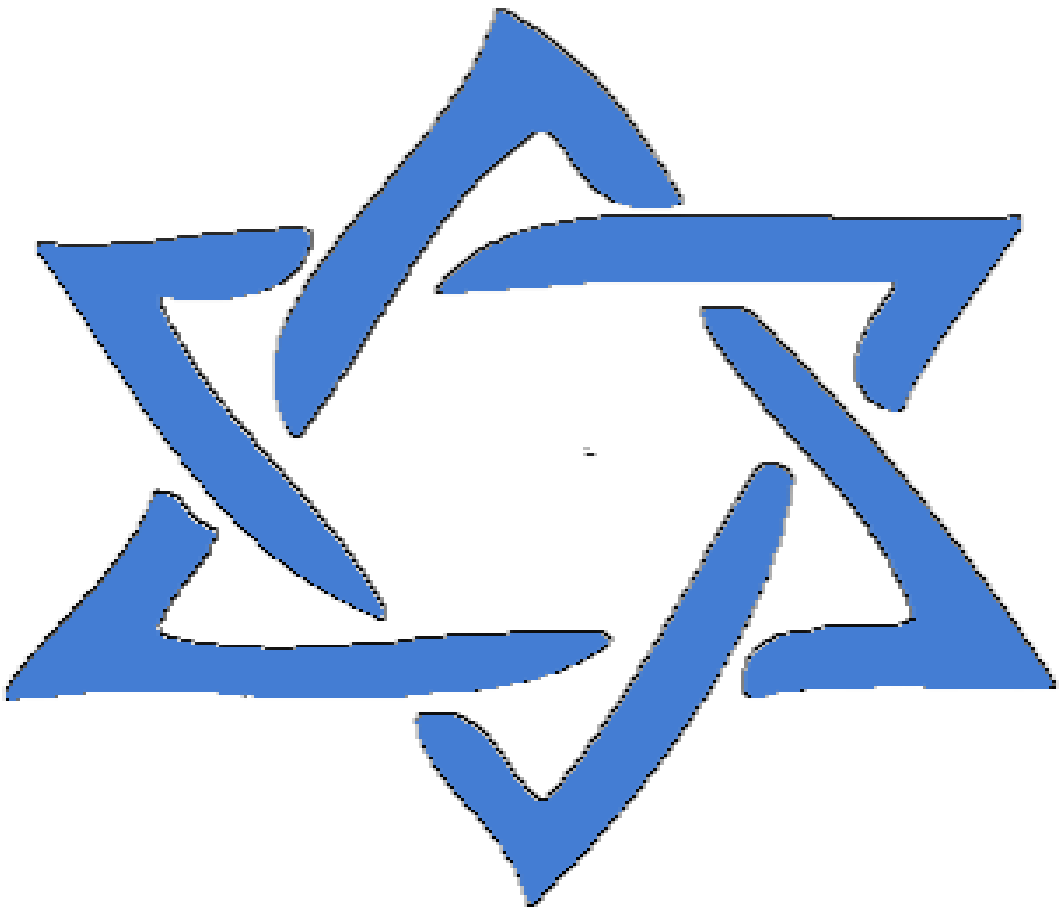
Binary search
Lesson 4



ACTIVITY SELECTION

Given N activities with their start and finish day given in array start[ ] and end[ ]. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a given day. Note : Duration of the activity includes both starting and ending day.

Input:

N = 2

start[] = {2, 1}

end[] = {2, 2}

Output: 1

Explanation:

A person can perform only one of the given activities.

ALGORITHM:

1) Sort the activities according to their finishing time and take a variable count=0.

2) Select the first activity from the sorted array and increase the count.

3) If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and increase the count...at end return count.

Time Complexity: O(n log n)

CODE:

```
int activitySelection(vector<int> start, vector<int> end, int n)
{
    vector<pair<int,int>>v;
    for(int i=0;i<n;i++)
    {
        v.push_back({start[i],end[i]});
    }
    sort(v.begin(),v.end(),
    [](pair<int,int>a,pair<int,int>b)
    {
        return a.second<b.second;
    });
    int count=1;
    pair<int,int>last=v[0];
    for(int i=1;i<n;i++)
    {
        if(v[i].first>last.second)
        {
            count++;
                last=v[i];
        }
         }
         return count;
     }
```

Que link: Activity Selection

AKKU AND BINARY NUMBERS

Akku likes binary numbers and she likes playing with these numbers. Her teacher once gave her a question.For given value of L and R, Akku have to find the count of number X, which have only three-set bits in it's binary representation such that "L ≤ X ≤ R".Akku is genius, she solved the problem easily. Can you do it ??

Input:

L = 11 and R = 19

Output: 4

Explanation:

There are 4 such numbers with 3 set bits in

range 11 to 19.

11 -> 1011

13 -> 1101

14 -> 1110

19 -> 10011

Expected Time Complexity: O(log(63^3))

Expected Auxiliary Space: O(1)

ALGORITHM:

If we can express a number into sum of powers of 2 ,e.g.

n=2^i+2^j+2^k where i not equals to j , not equals to k. Then the number n has 3 bits.

CODE:

```
vector<long long >v;

class Solution{
public:
    void precompute()
   {
      for(int i=0;i<63;i++)
      {
         for(int j=i+1;j<63;j++)
         {
            for(int k=j+1;k<63;k++)
            {
               v.push_back((pow(2,i))+(pow(2,j))+(pow(2,k)));
            }
         }
      }
      sort(v.begin(),v.end());
   }

      long long solve(long long l, long long r){
      // Code Here
```

```
        auto low=lower_bound(v.begin(),v.end(),l);
        auto high=upper_bound(v.begin(),v.end(),r);
        return (long long)(high-low);
    }
};

```

Que link: Akku and Binary Numbers

COIN PILES

There are N piles of coins each containing Ai (1<=i<=N) coins. Find the minimum number of coins to be removed such that the absolute difference of coins in any two piles is at most K.
Note: You can also remove a pile by removing all the coins of that pile.

Input:
N = 4, K = 0
arr[] = {2, 2, 2, 2}
Output:
0
Explanation:
For any two piles the difference in the number of coins is <=0. So no need to remove any coins.

ALGORITHM:
Naive approach : Since we cannot increase the number of coins in a pile. So, the minimum number of coins in any pile will remain the same as they can't be removed and increasing them will add to the operations which we need to minimize. Now, find the minimum coins in a pile and for every other pile if the difference between the coins in the current pile and the minimum coin pile is greater than K then remove the extra coins from the current pile.

Using binary search:
In order to find the minimum number of coins to be removed, we iterate over all the coin pile sizes,and for the rest of the piles if the pile size is less than the current size then remove that pile entirely otherwise if the

pile size is greater than current pile size plus K then remove the excess coins. The minimum number of coins removed in any of the turns is the answer.
Time Complexity: O(N*logN)
Auxiliary Space: O(N)

CODE:

```
int minSteps(int a[], int n, int k) {
 sort(a,a+n);
 vector<int>p(n);
 p[0]=a[0];
 for(int i=1;i<n;i++)
 {
    p[i]=p[i-1]+a[i];
 }
 int ans=INT_MAX,prev=0;
     for(int i=0;i<n;i++)
 {
         int pos=upper_bound(a+i,a+n,a[i]+k)-a;
         if(i>0 && a[i]!=a[i-1])
          prev=p[i-1];
          ans=min(ans,prev+p[n-1]-p[pos-1]-(n-pos)*(a[i]+k));
     }
     return ans;
  }
```

Que link: Coin Piles

THE PAINTER'S PARTITION PROBLEM

Dilpreet wants to paint his dog's home that has n boards with different lengths. The length of ith board is given by arr[i] where arr[] is an array of n integers. He hired k painters for this work and each painter takes 1 unit time to paint 1 unit of the board.

The problem is to find the minimum time to get this job done if all painters start together with the constraint that any painter will only paint continuous boards, say boards numbered {2,3,4} or only board {1} or nothing but not boards {2,4,5}.

Input:

n = 5

k = 3

arr[] = {5,10,30,20,15}

Output: 35

Explanation: The most optimal way will be:

Painter 1 allocation : {5,10}

Painter 2 allocation : {30}

Painter 3 allocation : {20,15}

Job will be done when all painters finish

i.e. at time = max(5+10, 30, 20+15) = 35

ALGORITHM:

We can see that the highest possible value in this range is the sum of all the elements in the array and this happens when we allot 1 painter all the sections of the board. The lowest possible value of this range is the maximum value of the array max, as in this allocation we can allot max to one painter and divide the other sections such that the cost of them is less than or equal to max and as close as possible to max. Now if we consider we use x painters in the above scenarios, it is obvious that as the value in the range increases, the value of x decreases and vice-versa. From this we can find the target value when x=k and use a helper function to find x, the minimum number of painters required when the maximum length of section a painter can paint is given.

The time complexity of the above approach is O(nlog(m)),where m is the sum of arr[i].

CODE:

```
long long minTime(int arr[], int n, int k)
  {
      long long max = 0;
      long long  sum=0;
      for (int i=0;i<n;i++)
      {
          sum=sum+arr[i];
          if(arr[i]>max)
          {
              max=arr[i];
          }
      }

      long long left = max;
      long long  right = sum;
```

```
    long long mid;
    long long answer;

    while(left<=right)
        {

            mid = (left + right)/2;
            int count =0;
            sum=0;
            bool flag= true;
            for(int i=0;i<n;i++)
            {
                sum= sum + arr[i];
                if(sum>mid)
                {
               count ++;
                    sum= arr[i];
                }
                if(count>=k)
                {
                flag = false;
                break;
                }
            }

        if(flag==true)
```

```
            {
                answer=mid;
                right= mid -1;
            }
            else
            {
                left = mid + 1;
            }
        }
    return answer;

    }
```

Que link: The Painter's Partition Problem

DISTRIBUTE N CANDIES AMONG K PEOPLE

Given N candies and K people. In the first turn, the first person gets 1 candy, the second gets 2 candies, and so on till K people. In the next turn, the first person gets K+1 candies, the second person gets K+2 candies and so on. If the number of candies is less than the required number of candies at every turn, then the person receives the remaining number of candies. Find the total number of candies every person has at the end.

Input:

N = 7, K = 4

Output:

1 2 3 1

Explanation:

At the first turn, the fourth person has to be given 4 candies, but there is only 1 left, hence he takes only one.

ALGORITHM:

NAIVE APPROACH:

Iterate to every turn and distribute candies, till the candies are finished.

Time Complexity : O(no. of distributions)

BETTER APPROACH:

1. Find such a value of n for which the sum of n positive numbers (n * (n+1)/2) is lesser than or equal to N (total no. of candies)

2. Find the number of complete rounds of candies (a round is said to be complete if all the K people get the desired number of candies in that turn and not less. Eg : 1, 2, 3, 4 is a complete round whereas 1, 2, 3, 2 is not)
    a. The complete rounds will be equal to n/K (n calculated in above step)
3. Find the number of extra distributions (i.e. no. of distributions left after completing the rounds)
    a. The extra distributions will be equal to (n%K + 1)
4. We can observe the i th person will have candies = (i + 1) + (i + 1 + K) + (i + 1 + 2*K) + (i + 1 + 3*K) + ………… upto number of complete rounds.

   = (i + 1)*rounds + (0 + 1 + 2 + ….. Upto number of rounds) *K

   = (i + 1) *rounds + (((rounds-1) * rounds)/2) * K
5. Iterate on K people and give them the calculated number of candies

   Additionally, provide the extra candies accordingly, starting from the first person.

Time Complexity: O(log N + K)

CODE:

```
long long BS( long long low, long long high, long long total)
{
   while(low<= high)
   {
      long long mid = low + (high - low)/2;
      if((mid * (mid + 1))/2 <= total)
         low = mid + 1;
```

```
     else
         high = mid - 1;
     }
return high;
 }

 vector<long long> distributeCandies(int N, int K)
 {

     vector<long long> sol (K, 0);
     if(K==1)
     {
         sol[0] = N;
         return sol;
     }

     long long low = 1, high = N + 1;
     long long count = BS(low, high, N);

     long long rounds = count/K;
     long long extras = count%K + 1;

     for(int i =0; i< K; i++)
     {
         if(i + 1 < extras)
             sol[i] += (rounds * K + (i + 1));
         //remaining candies after complete sum
         else if(i + 1 == extras)
             sol[i] += (N - (count*(count + 1))/2);

         //at any ith position candies
         //= (i+1) + (i+1+k) + (i+1+2k) +... so on upto rounds
         // = rounds*(i+1) + (1+2+3+.....(rounds-1))*k
         // = rounds*(i+1) + (rounds)*(rounds-1)*k

         sol[i] += (rounds*(i+1) +
                     ((rounds*(rounds-1))/2) *K );
     }
     return sol;
 }
```

Que link: [Distribute N candies among K people]()

CAPACITY TO SHIP PACKAGES WITHIN D DAYS

Given an array arr[] of N weights. Find the least weight capacity of a boat to ship all weights within D days.
Note: You have to load weights on the ship in the given order.

Input:
N = 3
arr[] = {1, 2, 1}
D = 2
Output:
3
Explanation:
We can ship the weights in 2 days in the following way:-
Day 1- 1,2
Day 2- 1

ALGORITHM:

1. The minimum capacity of the ship can be the maximum weight to be transported.
2. The maximum capacity of the ship can be the sum of all weights to be transported.
3. Now find the minimum of capacity for D days of transportation.

To do this:

I. Time complexity: **O ((max - min) * N)**

Iterate from min to max capacity, and find the number of days it takes to transport all weights. In every iteration, keep count of days :

a. Keep adding the capacity for each day, increment days if the capacity increases the iteration capacity.
b. If these days are not equal to D, increment the iteration capacity.
c. If it is equal to D, return Capacity.

II. Time Complexity **O (log(max - min) * N)**

Using Binary Search:

Instead of iterating to each capacity, find the mid element (i.e. min + (max - min)/2)

a. Count the number days as mentioned in linear approach.
b. If these days are greater than D, search in right part (i.e. low = mid + 1)
c. Else , search in left part (i.e. high = mid - 1) and keep track of ans variable
d. Lastly return ans

CODE:

```
int CountDays(int arr[], int N, int capacity)
{
   int days = 1;
   int cap_day = 0;
   //to keep track of capacity per day

```

```
for(int i =0; i<N; i++)
{
    cap_day += arr[i];
        if(cap_day > capacity)
    {
            days++;
            cap_day = arr[i];
        }
    }
    return days;
 }


 int leastWeightCapacity(int arr[], int N, int D)
 {
    // low is maximum of all the weights
    // high is sum of all weights
    int low = *max_element(arr, arr + N);
    int high = 0;
    for(int i = 0; i<N; i++)
        high += arr[i];


    //finding the days for corresponding
    //capacities and then comparing with D
    int ans = 0;
    while(low<= high)
    {
        int mid = low + (high - low)/2;

   //finding days required for min. capacity of mid
        int req_days=CountDays(arr,N,mid);

        if(req_days > D)
            low = mid + 1;
        else
        {
            ans = mid;
            high = mid - 1;
        }
    }
    return ans;
}
```

QUES LINK : [Capacity To Ship Packages Within D Days]()