GREEDY+HEAP

GREEDY:

Q1)maximum coins problem

```
int minCoins(vector<int>& coins, int amount) {
   sort(coins.begin(), coins.end());

   int res = 0;

   for (int i = coins.size() - 1; i >= 0; i--) {
      if (coins[i] <= amount) {
         int c = floor(amount / coins[i]);

         res += c;

         amount -= c * coins[i];
      }

      if (amount == 0) {
         break;
      }
   }

   return res;
}
```

2)activity selection problem

```
bool myCmp(pair<int, int> a, pair<int, int> b) {
   return (a.second < b.second);
}

int maxActivities(vector<pair<int, int>>& arr) {
   sort(arr.begin(), arr.end(), myCmp);

   int prev = 0;
   int res = 1;

   for (int curr = 1; curr < arr.size(); curr++) {
      if (arr[curr].first >= arr[prev].second) {
         res++;
         prev = curr;
      }
   }

   return res;
}
```

https://leetcode.com/problems/minimum-number-of-arrows-to-burst-balloons/

similar problem

3)Fractional knapsack problem

https://leetcode.com/problems/maximum-units-on-a-truck/

```
bool myCmp(pair<int, int> a, pair<int, int> b) {
   double r1 = (double)a.first / a.second;
   double r2 = (double)b.first / b.second;
   return r1 > r2;
}

double fKnapS(int W, vector<pair<int, int>>& arr) {
   sort(arr.begin(), arr.end(), myCmp);

   double res = 0.0;
```

```
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i].second <= W) {
            res += arr[i].first;
            W = W - arr[i].second;
        } else {
            res += arr[i].first * ((double) W / arr[i].second);
            break;
        }
    }

    return res;
}
```

Q4) JOB sequencing problem

Try it yourself

Q5)HUFF-MAN encoding

Try to explore both of the above conecepts

Q6) https://leetcode.com/problems/bag-of-tokens/

```
class Solution {
public:
    int bagOfTokensScore(vector<int>& tokens, int P) {
        int n = tokens.size();
        sort(tokens.begin(), tokens.end());

        int currScore = 0;
        int maxScore  = 0;
        int l = 0, r = n-1;

        //:GREEDY
        //While losing power, I will choose the smallest token
        //While gainin power, I will choose the largest token

        while(l <= r) {
            if(P >= tokens[l]) {
                currScore++;
                maxScore = max(maxScore, currScore); //keep updating it
                P -= tokens[l]; //choose smallest token
                l++;

            } else if(currScore >= 1) {
                currScore--;
                P += tokens[r]; //choose largest token
                r--;

            } else {
                //no way further to increase score
                return maxScore;
            }
        }
        return maxScore;
    }
};
```

Q7) https://leetcode.com/problems/boats-to-save-people/description/

```
class Solution {
public:
  int numRescueBoats(vector<int>& people, int limit) {
      sort(people.begin(),people.end());
      int n=people.size();
      int i=0,j=n-1;
      int count=0;
      while(i<=j){
        int sum=people[i]+people[j];
        if(sum>limit){
          count++;
          j--;
        }else{
          count++;
          i++;
          j--;
        }

      }
      return count;
  }
};
```

Q8) https://leetcode.com/problems/break-a-palindrome/

```
class Solution {
public:
  string breakPalindrome(string pal) {
      int n = pal.length();
      if(n == 1) return "";

      for(int i = 0; i<n/2; i++) {
        if(pal[i] != 'a') {
          pal[i] = 'a';
          return pal;
        }
      }

      pal[n-1] = 'b';
      return pal;
  }
};
```

Q9) https://leetcode.com/problems/broken-calculator/description/

```
class Solution {
public:
  int brokenCalc(int startValue, int target) {
      if(startValue >= target)
        return startValue-target;

      //even
      if(target%2 == 0) {
        return 1 + brokenCalc(startValue, target/2);
      }



      return 1 + brokenCalc(startValue, target+1);
  }
};
```

Q10) https://leetcode.com/problems/minimum-time-to-make-rope-colorful/

```
class Solution {
public:
   int minCost(string colors, vector<int>&
neededTime) {
        int n = colors.size();

        int time    = 0;
        int prevMax = 0;

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

            if(i > 0 && colors[i] != colors[i-1]) {
                prevMax = 0;
            }

            int curr = neededTime[i];

            time += min(prevMax, curr); //greedily

            prevMax = max(prevMax, curr);

        }

        return time;
    }
};
```

Q11) https://leetcode.com/problems/earliest-possible-day-of-full-bloom/description/

```
class Solution {
public:
   int earliestFullBloom(vector<int>&
plantTime, vector<int>& growTime) {
        int n = plantTime.size();

        vector<pair<int, int>> vec(n);
//{plantTime[i], growTime[i]}

        for(int i = 0; i<n; i++) {
            vec[i] = {plantTime[i], growTime[i]};
        }

         // sort the grow_plant_times of seeds by
their growTime in descending order.
        // It makes sense to plant the seed with
maximum growTime first

        //sort according to grow time (in
descending order)
        auto lambda = [](pair<int, int>& P1,
pair<int, int>& P2) {
            return P1.second > P2.second;
        };

        sort(vec.begin(), vec.end(), lambda);

        // a seed is planted only after the seeds
having greater bloom time than the current
seed are planted.
        // So essentially, the plant time of a seed
is the sum of plant times of all the seeds
preceeding this seed
        // and the plant time of the seed itself
        // we store the plant time of preceeding
seeds in the following variable, prevPlantDays
```

```
        int prevPlantDays = 0;

        int maxBloomDays  = 0;
        for(int i = 0; i<n; i++) {
            int currPlantTime = vec[i].first;
            int currGrowTime  = vec[i].second;

            // adding the plant time of ith seed to the plant times of preceeding seeds
            // it would take prev_plant_time amount of time to actually plant the ith seed
            prevPlantDays += currPlantTime;


            // bloom time of ith seed = total plant time of ith seed + grow time of ith seed + 1
            // (as the flower blooms after last day of it's growth)
            int currPlantBloomTime = prevPlantDays + currGrowTime;

            maxBloomDays = max(maxBloomDays, currPlantBloomTime);
        }

        return maxBloomDays;

    }
};
```

Q12) https://leetcode.com/problems/longest-palindrome-by-concatenating-two-letter-words/description/

```
class Solution {
public:
    int longestPalindrome(vector<string>& words) {
        unordered_map<string, int> mp;


        //update map with frequency
        for(string &word : words) {
            mp[word]++;
        }

        bool centerUsed = false; //for frequency one waale strings

        int result = 0;

        //start iterating on words one by one
        for(string &word : words) {
            string rev = word;
            reverse(begin(rev), end(rev));

            if(rev != word) { //"ab" "ba"    ->.  {ab ba}
                if(mp[word] > 0 && mp[rev] > 0) {
                    mp[word]--;
                    mp[rev]--;
                    result += 4;
                }
            } else { //"abcc     ccba"
            if(mp[word] >= 2) {
                mp[word] -= 2;
                result += 4;
            } else if(mp[word] == 1 &&
centerUsed == false) {
                mp[word]--;
                result += 2;
                centerUsed = true; //ab use
hogaya hai center wala
            }
        }
    }

    return result;
  }
};
```

Q13) https://leetcode.com/problems/maximum-69-number/

```
class Solution {
public:
  int maximum69Number (int num) {
    int place = 0;
    int index = -1;
    int temp = num;
    while(temp) {
      int remain = temp%10;
      if(remain == 6)
        index = place;

      temp = temp/10;
      place++;
    }

    if(index == -1) return num;

    return num + 3*pow(10, index);

  }
};
```

Second approach:

```
class Solution {
public:
  int maximum69Number(int num) {
    string numStr = to_string(num);

    for(int i = 0; i < numStr.size(); i++) {
      if(numStr[i] == '6') {
        numStr[i] = '9';
        break;
      }
    }

    return stoi(numStr);
  }
};
```

Q14) https://leetcode.com/problems/maximum-bags-with-full-capacity-of-rocks/description/

```
class Solution {
public:
    int maximumBags(vector<int>& capacity, vector<int>& rocks, int additionalRocks) {
        int n = capacity.size();
        int count = 0;
        vector<int> vec(n);

        for(int i = 0; i<n; i++) {
            vec[i] = capacity[i] - rocks[i];
        }

        sort(begin(vec), end(vec));

        for(int i = 0; i<n; i++) {
            if(additionalRocks >= vec[i]) {
                additionalRocks -= vec[i];
                count++;
            } else
                break;
        }

        return count;
    }
};
```

Q15)

https://leetcode.com/problems/minimum-rounds-to-complete-all-tasks/

```
class Solution {
public:
    int minimumRounds(vector<int>& tasks) {
        unordered_map<int, int> mp;

        for(int &x : tasks) {
            mp[x]++;
        }

        int round = 0;

        for(auto &it : mp) {
            int count = it.second;

            if(count == 1)
                return -1;

            if(count % 3 == 0)
                round += count/3;
            else
                round += count/3+1;
        }

        return round;
    }
};
```

16) https://leetcode.com/problems/maximum-ice-cream-bars/

```
class Solution {
public:
    int maxIceCream(vector<int>& costs, int coins) {
        sort(begin(costs), end(costs));

        int count = 0;

        for(int &cost : costs) {
            if(cost > coins)
                return count;
            else {
                count++;
                coins -= cost;
            }
        }

        return count;
    }
};
```

Q17) https://leetcode.com/problems/optimal-partition-of-string/

```
class Solution {
public:
    int partitionString(string s) {
        vector<int> lastSeen(26, -1);

        int count = 0;
        int substringStart = 0;

        for (int i = 0; i < s.length(); i++) {
            if (lastSeen[s[i] - 'a'] >= substringStart) {
                count++;
                substringStart = i;
            }
            lastSeen[s[i] - 'a'] = i;
        }

        return count+1;
    }
};
```

Second way:

```
class Solution {
public:
    int partitionString(string s) {
        unordered_set<char> seen;
        int count = 1;  // Start with 1 because there is at least one substring

        for (char c : s) {
            if (seen.count(c)) {
                count++;
                seen.clear();
            }
            seen.insert(c);
        }

        return count;
    }
};
```

Q18) https://leetcode.com/problems/dota2-senate/description/

```
class Solution {
public:

    bool removeSenator(string &senate, char ch, int idx) {
        bool loopAround = false;

        while(true) {

            if(idx == 0) {
                loopAround = true;
            }

            if(senate[idx] == ch) {
                senate.erase(begin(senate) + idx);
                break;
            }

            idx = (idx+1)%(senate.length());

        }

        return loopAround;
    }

    string predictPartyVictory(string senate) {

        int R_Count = count(begin(senate),
end(senate), 'R');
        int D_Count = senate.length() - R_Count;


        int idx = 0;

        while(R_Count > 0 && D_Count > 0) {

            if(senate[idx] == 'R') {
                bool checkRemoval =
removeSenator(senate, 'D',
(idx+1)%(senate.length()));
                D_Count--;
                if(checkRemoval) {
                    idx--;
                }
            } else {
                bool checkRemoval =
removeSenator(senate, 'R',
(idx+1)%(senate.length()));
                R_Count--;
                if(checkRemoval) {
                    idx--;
                }
            }

            idx = (idx+1)%(senate.length());

        }

        return R_Count == 0 ? "Dire" : "Radiant";

    }
};
```

Method -2:

https://leetcode.com/problems/dota2-senate/solutions/3483399/simple-diagram-explanation/

better explanation using queues that was brute force what we did but queue method analayse

Q19) https://leetcode.com/problems/minimum-replacements-to-sort-the-array/description/

```
class Solution {
public:
    long long
minimumReplacement(vector<int>& nums) {
        int n = nums.size();
        long long result = 0;

        for(int i = n-2; i >= 0; i--) {

            int splits = nums[i]/nums[i+1];

            if((nums[i]) % nums[i+1] != 0) {
                splits++;
            }

            nums[i] = nums[i]/splits;
            result += splits-1;
        }

        return result;
    }
};
```

Q20)

https://leetcode.com/problems/minimum-deletions-to-make-character-frequencies-unique/description/

```
class Solution {
public:
    int minDeletions(string s) {
        unordered_set<int> st;
        int freq[26] = {0};

        for(char &ch : s) {
            freq[ch-'a']++;
        }
        int result = 0;
        for(int i = 0; i<26; i++) {

            while(freq[i] > 0 && st.find(freq[i]) != st.end()) {
                freq[i]--;
                result++;
            }
            st.insert(freq[i]);
        }

        return result;
    }
};
```

Q21)

```
class Solution {
public:
    bool winnerOfGame(string colors) {
        int n = colors.length();
        int alice = 0;
        int bob   = 0;

        for(int i = 1; i<n; i++) {

            if(colors[i-1] == colors[i] && colors[i] == colors[i+1]) {
                if(colors[i] == 'A')
                    alice++;
                else
                    bob++;
            }
        }
        return alice > bob;
    }
};
```

Q22) https://leetcode.com/problems/maximum-score-of-a-good-subarray/

```
class Solution {
public:
    int maximumScore(vector<int>& nums, int k) {
        int n = nums.size();

        int i = k;
        int j = k;
        int curMin = nums[k];

        int result = nums[k];

        while(i > 0 || j < n-1) {
            int leftValue;
            int rightValue;

            if(i > 0) {
                leftValue = nums[i-1];
            } else {
                leftValue = 0;
            }

            if(j < n-1) {
                rightValue = nums[j+1];
            } else {
                rightValue = 0;
            }

            if(leftValue < rightValue) {
                j++;
                curMin = min(curMin, nums[j]);
            } else {
                i--;
                curMin = min(curMin, nums[i]);
            }

            result = max(result, curMin * (j-i+1));
        }

        return result;
    }
};
```

HEAP

Q1)BASICs of priority queue

By default it's a max heap

```
int main(){
   priority_queue <int> pq;
   pq.push(10);
   pq.push(15);
   pq.push(5);
   cout<<pq.size()<<" ";
   cout<<pq.top()<<" ";
   while(pq.empty()==false){
      cout<<pq.top()<<" ";
      pq.pop();
   }

   return 0;
}
```

For min heap

```
int main(){
   priority_queue
<int,vector<int>,greater<int>> pq;
   pq.push(10);
   pq.push(15);
   pq.push(5);
   cout<<pq.size()<<" ";
   cout<<pq.top()<<" ";
   while(pq.empty()==false){
      cout<<pq.top()<<" ";
      pq.pop();
   }
   return 0;
}
```

Q2)sort a k sorted array

```
void sortK(vector<int>& arr, int k){
   priority_queue<int, vector<int>,
greater<int> > pq;

   for(int i = 0; i <= k && i < arr.size(); i++)
      pq.push(arr[i]);

   int index = 0;
   for (int i = k + 1; i < arr.size(); i++) {
      arr[index++] = pq.top();
      pq.pop();
      pq.push(arr[i]);
   }

   while (!pq.empty()) {
      arr[index++] = pq.top();
      pq.pop();
   }
}
```

Q3) https://leetcode.com/problems/find-median-from-data-stream/

```
class MedianFinder {
public:
   priority_queue<int> left_max_heap; //max
heap
   priority_queue<int, vector<int>,
greater<int>> right_min_heap; //min heap
   MedianFinder() {
    }

    void addNum(int num) {
        if(left_max_heap.empty() || num < left_max_heap.top()) {
            left_max_heap.push(num);
        } else {
            right_min_heap.push(num);
        }

        //always maintain left_max_heap size one greater than rigfht_min_heap size
        //ya fir, dono ka size equal ho

        if(abs((int)left_max_heap.size() - (int)right_min_heap.size()) > 1) {
            right_min_heap.push(left_max_heap.top());
            left_max_heap.pop();
        } else if(left_max_heap.size() < right_min_heap.size()) {
            left_max_heap.push(right_min_heap.top());
            right_min_heap.pop();
        }

    }

    double findMedian() {
        if(left_max_heap.size() == right_min_heap.size()) {
            // matlab even number of elements hue honge
            return (double)(left_max_heap.top()+right_min_heap.top())/2;
        }

        //else hamare paas odd number of elemenes hue honge
        return left_max_heap.top();
    }
};
```

Q4) https://leetcode.com/problems/sort-characters-by-frequency/

```
class Solution {
public:
    typedef pair<char, int> P;
    struct comp {
        bool operator()(P &p1, P &p2) {
            return p1.second<p2.second; //max-heap
        }
    };

    string frequencySort(string s) {
        priority_queue<P, vector<P>, comp> pq;

        unordered_map<char, int> mp;
        for(char &ch : s) {
            mp[ch]++;
        }

        for(auto &it : mp) {
            pq.push({it.first, it.second});
        }

        string result = "";

        while(!pq.empty()) {
            P temp = pq.top();
            pq.pop();

            result += string(temp.second,
temp.first);
        }
        return result;
    }
};
```

Q5) https://leetcode.com/problems/remove-stones-to-minimize-the-total/description/

```
class Solution {
public:
    int minStoneSum(vector<int>& piles, int k) {
        int n = piles.size();

        priority_queue<int> pq;
        int sum = 0;
        for(int i = 0 ; i<n; i++) {
            pq.push(piles[i]);
            sum += piles[i];
        }

       for(int i = 1; i<=k; i++) {
           int curr = pq.top();
           pq.pop();

           int remove = curr/2;
           sum -= remove;

           int remaining = curr-remove;

           pq.push(remaining);
       }

       return sum;

    }
};
```

Q6) https://leetcode.com/problems/single-threaded-cpu/description/

```
class Solution {
public:
    vector<int> getOrder(vector<vector<int>>&
tasks) {
        int n = tasks.size();

        vector<array<int, 3>> sortedTasks;

        for(int i = 0; i<n; i++) {
            int start_time = tasks[i][0];
            int processing_time = tasks[i][1];

            sortedTasks.push_back({start_time,
processing_time, i});
        }
        //sort it

        sort(begin(sortedTasks),
end(sortedTasks)); //O(nlogn)


        vector<int> result;


        long long curr_time = 0;

        int idx            = 0;


        priority_queue< pair<int, int>,
vector<pair<int, int>>, greater<> > pq;
//min_heap


        while(idx < n || !pq.empty()) {


            if(pq.empty() && curr_time <
sortedTasks[idx][0]) {

                curr_time = sortedTasks[idx][0];

            }



            while(idx < n && sortedTasks[idx][0] <=
curr_time) {

                pq.push({sortedTasks[idx][1],
sortedTasks[idx][2]}); //log(n)

                idx++;

            }


            pair<int, int> curr_task = pq.top();

            pq.pop();


            curr_time += curr_task.first;
//processing time

                result.push_back(curr_task.second);

            }


            o(nlogn)

            return result;

        }

};
```

Q7) https://leetcode.com/problems/last-stone-weight/

```
class Solution {

public:

    int lastStoneWeight(vector<int>& stones) {

        priority_queue<int> pq; //max-heap

        for(int i:stones)

            pq.push(i);

        while(pq.size() > 1) {

            int a = pq.top();

            pq.pop();

            int b = pq.top();

            pq.pop();

            if(a != b)

                pq.push(abs(a-b));

        }


        if(pq.size())

            return pq.top();

        return 0;

    }

};
```

Q8) https://leetcode.com/problems/top-k-frequent-elements/description/

```
class Solution {
public:
   typedef pair<int, int> p;

   vector<int> topKFrequent(vector<int>& nums, int k) {

      //min-heap
      priority_queue<p, vector<p>, greater<p>> pq;

      //count frequency of each element
      // Worst Case - n distinct elements are stored, so, space O(n)
      unordered_map<int, int> mp;
      for(int i : nums)
         mp[i]++;

      //Push in min-heap and maintain size k
      for(auto it:mp) {
         pq.push({it.second, it.first});

         if(pq.size() > k)
            pq.pop();
      }


      //Pick all top K elements
      vector<int> result;
      while(!pq.empty()) {
         result.push_back(pq.top().second);
         pq.pop();
      }
      return result;
   }
};
```

Approach 2 (use of bucket sort)

Check out yourself

Q9) **https://leetcode.com/problems/kth-largest-element-in-a-stream/**

```
class KthLargest {
public:
   priority_queue<int, vector<int>, greater<int>> pq;
   int K;

   KthLargest(int k, vector<int>& nums) {
      K = k;
      for(int &x : nums) {
         pq.push(x);

         if(pq.size() > k)
            pq.pop();
      }
   }

   int add(int val) {
      pq.push(val);

      if(pq.size() > K)
         pq.pop();

      return pq.top();
   }
};
```

Q10) https://leetcode.com/problems/maximum-subsequence-score/description/

DP:

```
class Solution {
public:
    int K;
    int n;
    unordered_map<string, int> mp;

    long long solve(vector<int>& nums1, vector<int>& nums2, int sum, int min_el, int i, int count) {
        if(count == K) {
            return sum * min_el;
        }
         if(i >= n) {
            return 0;
        }

        string key = to_string(sum) + "_" + to_string(min_el) + "_" + to_string(i) + "_" + to_string(count);
        if(mp.find(key) != mp.end())
            return mp[key];

        int min_now = min(min_el, nums2[i]);

        long take_i = solve(nums1 , nums2 , sum + nums1[i] , min_now, i+1 , count+1);
        long not_take_i = solve(nums1 , nums2 , sum, min_el, i+1 , count);

        return mp[key] = max(take_i, not_take_i);
    }

    long long maxScore(vector<int>& nums1, vector<int>& nums2, int k) {
        K = k;
        n = nums1.size();
        mp.clear();

        return solve(nums1 , nums2 , 0 , INT_MAX , 0 , 0);
    }
};
```

PRIORITY QUEUE:

```
class Solution {
public:
    long long maxScore(vector<int>& nums1, vector<int>& nums2, int k) {
        int n = nums1.size();

        vector<pair<int,int>> vec(n);

        for(int i = 0; i<n; i++) {
            vec[i] = {nums1[i], nums2[i]};
        }

        auto lambda = [&](auto &P1, auto &P2) {
            return P1.second > P2.second;
        };
        sort(begin(vec), end(vec), lambda);

        priority_queue<int, vector<int>,
greater<int>> pq; //min_heap

        long long Ksum = 0;

        for(int i = 0; i<=k-1; i++) {

            Ksum += vec[i].first;
            pq.push(vec[i].first);

        }

        long long result = Ksum * vec[k-1].second;

        for(int i = k; i<n; i++) {

            //taking minimum as vec[i].second
            Ksum += vec[i].first - pq.top();
            pq.pop();

            pq.push(vec[i].first);

            result = max(result, Ksum *
vec[i].second);

        }

        return result;
    }
};
```

Q11) https://leetcode.com/problems/total-cost-to-hire-k-workers/

```
class Solution {
public:
    long long totalCost(vector<int>& costs, int
k, int candidates) {
        int n = costs.size();

        priority_queue<int,vector<int>,greater<in
t>> pq1,pq2;

        long long ans = 0;

        int hired = 0;
        int i = 0;
        int j = n-1;

        while(hired < k){

            while(pq1.size() < candidates && i<=j)
                pq1.push(costs[i++]);
            while(pq2.size()<candidates && j>=i)
                pq2.push(costs[j--]);

            int a = pq1.size() > 0 ? pq1.top() :
INT_MAX;
            int b = pq2.size() > 0 ? pq2.top() :
INT_MAX;

            if(a <= b){
                ans += a;
                pq1.pop();
```

```
          } else {
              ans += b;
              pq2.pop();
          }

          hired++;
      }
      return ans;
  }
};
```

Think of how to solve using only 1 heap

Q12) **https://leetcode.com/problems/kth-largest-element-in-an-array/**

```
class Solution {
public:
  int findKthLargest(vector<int>& nums, int k)
{
      priority_queue<int, vector<int>,
greater<int>> minh;
      for(int i:nums) {
          minh.push(i);
          if(minh.size() > k)
              minh.pop();
      }

      return minh.top();
  }
};
```

Q13) **https://leetcode.com/problems/task-scheduler/**

```
class Solution {
public:
  int leastInterval(vector<char>& tasks, int p)
{
      int n = tasks.size();
      unordered_map<char, int> mp;

      for(char &ch : tasks) {
          mp[ch]++;
      }

      priority_queue<int> pq; //max heap
      //we want to finish the process which is
most occurring (having highest frequency)
      //so that we don't have to finish in the
last with p gaps.
      int time = 0;

      for(auto &it : mp) {
          pq.push(it.second);
      }

      while(!pq.empty()) {
          vector<int> temp;
          for(int i = 1; i<=p+1; i++) {
              //filling first p+1 characters
              if(!pq.empty()) {
                  temp.push_back(pq.top()-1);
//finishing one instance of each process
                  pq.pop();
              }
          }
```

```
            for(int &freq : temp) {
                if(freq > 0)
                    pq.push(freq);
            }

            if(pq.empty()) //all processes finished
                time += temp.size();
            else
                time += (p+1); //we finished p+1
tasks above in the loop

        }

        return time;
    }
};
```

Try using greedy