

## Must Do Greedy Algorithm problems for Placement

<b>Greedy</b>	<a href="#"><u>Activity Selection Problem</u></a>
<b>Greedy</b>	<a href="#"><u>Job Sequencing Problem</u></a>
<b>Greedy</b>	<a href="#"><u>Huffman Coding</u></a>
<b>Greedy</b>	<a href="#"><u>Water Connection Problem</u></a>
<b>Greedy</b>	<a href="#"><u>Fractional Knapsack Problem</u></a>
<b>Greedy</b>	<a href="#"><u>Greedy Algorithm to find Minimum number of Coins</u></a>
<b>Greedy</b>	<a href="#"><u>Maximum trains for which stoppage can be provided</u></a>
<b>Greedy</b>	<a href="#"><u>Minimum Platforms Problem</u></a>
<b>Greedy</b>	<a href="#"><u>Buy Maximum Stocks if i stocks can be bought on i-th day</u></a>
<b>Greedy</b>	<a href="#"><u>Find the minimum and maximum amount to buy all N candies</u></a>
<b>Greedy</b>	<a href="#"><u>Minimize Cash Flow among a given set of friends who have borrowed money from each other</u></a>
<b>Greedy</b>	<a href="#"><u>Minimum Cost to cut a board into squares</u></a>
<b>Greedy</b>	<a href="#"><u>Check if it is possible to survive on Island</u></a>
<b>Greedy</b>	<a href="#"><u>Find maximum meetings in one room</u></a>
<b>Greedy</b>	<a href="#"><u>Maximum product subset of an array</u></a>
<b>Greedy</b>	<a href="#"><u>Maximize array sum after K negations</u></a>
<b>Greedy</b>	<a href="#"><u>Maximize the sum of arr[i]*i</u></a>
<b>Greedy</b>	<a href="#"><u>Maximum sum of absolute difference of an array</u></a>
<b>Greedy</b>	<a href="#"><u>Maximize sum of consecutive differences in a circular array</u></a>
<b>Greedy</b>	<a href="#"><u>Minimum sum of absolute difference of pairs of two arrays</u></a>
<b>Greedy</b>	<a href="#"><u>Program for Shortest Job First (or SJF) CPU Scheduling</u></a>
<b>Greedy</b>	<a href="#"><u>Program for Least Recently Used (LRU) Page Replacement algorithm</u></a>
<b>Greedy</b>	<a href="#"><u>Smallest subset with sum greater than all other elements</u></a>
<b>Greedy</b>	<a href="#"><u>Chocolate Distribution Problem</u></a>
<b>Greedy</b>	<a href="#"><u>DEFKIN -Defense of a Kingdom</u></a>
<b>Greedy</b>	<a href="#"><u>DIEHARD -DIE HARD</u></a>
<b>Greedy</b>	<a href="#"><u>GERGOVIA -Wine trading in Gergovia</u></a>
<b>Greedy</b>	<a href="#"><u>Picking Up Chicks</u></a>
<b>Greedy</b>	<a href="#"><u>CHOCOLA –Chocolate</u></a>

<b>Greedy</b>	<a href="#"><u>ARRANGE -Arranging Amplifiers</u></a>
<b>Greedy</b>	<a href="#"><u>K Centers Problem</u></a>
<b>Greedy</b>	<a href="#"><u>Minimum Cost of ropes</u></a>
<b>Greedy</b>	<a href="#"><u>Find smallest number with given number of digits and sum of digits</u></a>
<b>Greedy</b>	<a href="#"><u>Rearrange characters in a string such that no two adjacent are same</u></a>
<b>Greedy</b>	<a href="#"><u>Find maximum sum possible equal sum of three stacks</u></a>

# Greedy Algorithm

Date 3/4/22 Page no. \_\_\_\_\_

Ques  $\rightarrow$  N meetings in one room (Activity Selection prob)

There is a meeting room in a firm. There are N meetings in form of  $(start[i], end[i])$  where  $start[i]$  is start time of meet i, &  $end[i]$  is finish time of meeting 'i'

What is the maximum no. of meeting that can be accomplished, when one meeting held in meet room at a particular time?

\* Example :-

$$S[] = \{1, 3, 0, 5, 8, 5\} \text{ Ans } \rightarrow 4$$

$$F[] = \{2, 4, 6, 7, 9, 9\} \quad (1, 2), (3, 4), (5, 7), (8, 9)$$

\* Imp \* whenever a situation comes where we've to find max. from the given sequence then, there is greedy approach comes into play.)

\* Intuition :-

$\therefore$  we've to find maxi from given sequence  
we'll apply sorting,  
we can sort start & finish time, 2 type  
of sorting start time and end time sorting

In this case we'll use end time sorting

$$(1, 2) \quad (3, 4) \quad (0, 6) \quad (5, 7) \quad (5, 9) \quad (8, 9)$$

Here end is same so small ones will be considered.

### \* Approach :-

- ① Select all pair in ~~using~~ order of finish time in each pair.
- ② Select first meeting of sorted pair as the first meet in room & push into result vector & set variable say time\_limit with second value ( $F[1]$ ) of first selected meet.
- ③ Iterate from the second pair to last pair of array and if value of first or start time of current pair is greater than previously selected pair finish time (time\_limit) then select the current pair and update result vector (push selected meet no. into vector) & var time\_limit
- ④ print order of meeting from vector.

### \* Code :-

```
for( i=1; i<n; i++)
```

```
    if ( srt[i].second ] > time_limit )
```

```
        m.push_back ( a[i].second + 1 );
```

```
        time_limit < a[i].first ;
```

(3)

(3)

## Ques - Job Sequencing Problem

Given an array of job where every job has a deadline and associated profit if the job is finished before the deadline. It is given that job takes a single unit of time, so the min. possible deadline for any job is 1.

### Example :-

Input :-	JobID	a	b	c	d
	Deadline	4	1	1	1
	Profit	20	10	40	30

(Output :- Maximum profit sequence : [c, a])

### \* Intuition :-

generate all subsets of given set of job and check individual subset for the feasibility of job in that subset.

keep the track of maximum profit among all feasible subset.

(This is standard greedy approach.)

Tc -  $O(n^2)$

We can optimize using priority queue (max-heap)

## ② Algorithm :-

sort the job based on their deadlines  
 Iterate from end and calculate  
 the available slot between every two  
 consecutive deadlines. Include the profit,  
 deadline, and job ID of ith job in the  
 max heap.

while the slot are available and  
 there are jobs left in max heap, include  
 the job ID with max. profit and deadline  
 in the result.

Sort the result array based on their  
 deadlines.

## ③ Code :-

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

```
  for (int j = min(n, arr[i].dead) - 1;  
       j >= 0; j--)
```

```
    if (slot[j] == false)
```

```
      result[j] = i;
```

```
      slot[j] = true;
```

```
      break;
```

③

③

## Ques → Huffman Coding

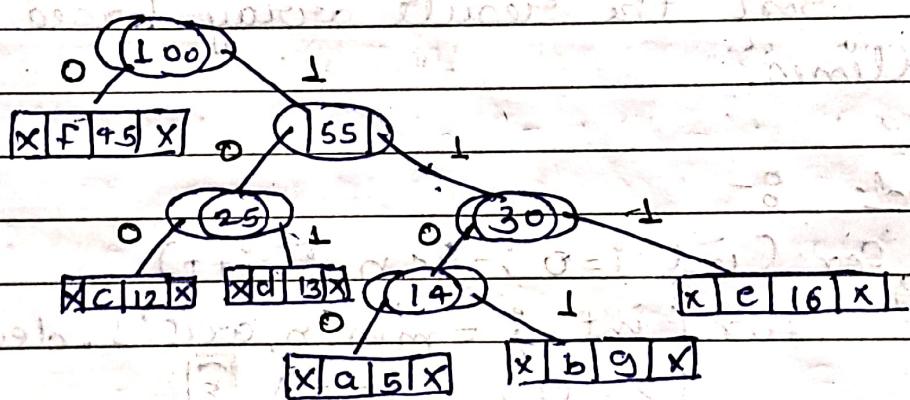
Given string  $S$  of distinct character of size  $N$  and their corresponding frequency  $f[i]$ .  
i.e. character  $S[i]$  has  $f[i]$  frequency.

Task is to build the huffman tree  
print all the huffman codes in preorder traversal of the tree.

Example :-

$S = 'abcdef'$        $f[.] = \{5, 9, 12, 13, 16, 45\}$

O/p: 0, 100, 101, 1100, 1101, 111



Huffman Code will be :-

$f : 0, c = 100, d = 101, a = 1100, b = 1101, e = 111$

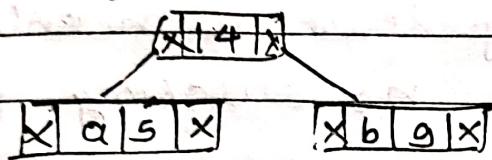
(print them in preorder traversal)

NOTE:

It is use in case where there is series of frequently occurring character.

### Steps to create (min heap) Huffman tree

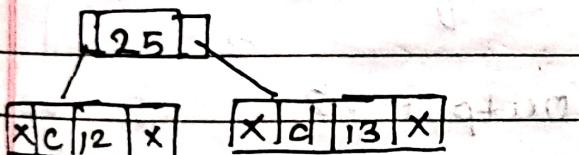
- ① Build a min heap that contain 6 nodes where each node represent root of tree with single node.
- ② Extract 2 min freq. node from min. heap. Add a new internal node with freq.  $5 + 9 = 14$



Now min heap contains 5 node where 4 nodes are root of tree with single element each and 1 heap node's root with 3 elements.

character	c	d	Internal node	e	f
frequency	12	13		16	45

- ③ Extract two min. freq. nodes from heap. add a new internal node with freq.  $12 + 13 = 25$ .



- ④ Extract 2 min freq. node. Add new internal node with freq.  $14 + 16 = 30$

- ⑤ Extract 2 min freq. node. Add a new internal node with freq.  $25 + 30 = 55$

- ⑥ Extract 2 min freq. node. Add new internal node with freq.  $45 + 55 = 100$

### Ques → Water Connection Problem

Every house in colony has almost 1 pipe going into the house and 1 pipe almost going out of it. Tanks and taps are to be installed in a manner such that every house with one outgoing pipe but no incoming pipe gets a tank installed on its roof and every house with only one incoming pipe and no outgoing pipe gets a tap.

Given 2 integers n and p denoting no. of houses & the number of pipes. the connection of pipe among the houses contain 3 input value :  $a_i$ ,  $b_i$ ,  $d_i$ .  $a_i$  denotes diameter of pipe of house  $b_i$ . from  $a_i$  to house  $b_i$  find out the efficient no. of tanks for the network

#### \* Example

4 3  
1 2 60

Output : 8 1 2 60  
3 4 50

3 4 50  
4 4 80

connected components

are of  $1 \rightarrow 2$  &  $3 \rightarrow 4$

∴, our ans is 2 follow by

1 2 60 and 3 4 50

## \* Intuition

- Perform DFS from appropriate houses to find all different connected components. The no. of diff. connected components in our answer T.
- The next t line of output are the beginning of connected components, end of the connected components and the memo diameter from start to end of connected components in each line.
- °°, tank can be installed only on the house having outgoing pipe and no incoming pipe,
- °°, these are appropriate house to start DFS from i.e. perform DFS from such unvisited houses.

## Ours → Fractional Knapsack

Given weights and values of  $N$  items, we need to put these items in a knapsack of capacity  $W$  to get maximum total value in the knapsack.

### Example

$N = 3$ ,  $W = 50$  output :-  
 $\text{value}[] = \{60, 100, 120\}$  240.00  
 $\text{weight}[] = \{10, 20, 30\}$  (total maximum value  
 and total weight of items we can have is 240.00  
 from given capacity of sack).

Intuition :- (greedy approach)

Idea is to calculate ratio value/weight for each item & sort on the basis of this ratio.

Then take the items with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can, which will always be optimal soln to problem.

## Implementation

```
double fractionalknapsack(int w,
                           struct Item arr, int n).
```

F

```
sort(arr, arr+n, cmp);
```

```
int currlight = 0;
```

```
double finalValue = 0.0;
```

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

if (currlight + arr[i].weight <= w)

Σ

```
    currlight += arr[i].weight;
```

```
    finalValue += arr[i].value;
```

③

Σ

else

```
    int remain = w - currlight;
```

```
    finalValue += arr[i].value
```

\* (double) remain /

(double) arr[i].weight);

③

③

3

Ques → Greedy algo to find min. no. of coins  
(choose and swap - gfg).

You are given a string  $s$  of lowercase case. You can choose any two char in the string & replace all the occurrence of first char with second char and vice-versa. find lexicographically smallest string that can be obtained by doing this operation at most once.

### ① Example

$A = "ccad"$  Output = "aacd"

In ccad - we choose a and c & after doing replacement opn once we get aacd.

### ② Intuition

→ firstly store whole string "cccad", when we pick 'c' to we check whether smaller than c exist in string or not, if yes i.e a, replace a and c  $\Rightarrow$  "a~~a~~c~~d~~d".

→ string - dcba  $\rightarrow$  find smaller than d i.e a, b, c  
∴ a is  $\uparrow$  smallest  $\therefore$ , d replace by a  
"acbd"

→ String adcba = a it self is smallest  
so  $\uparrow\uparrow$  no replacement.

Now at d  $\rightarrow$  a is smallest but a already checked for an 2<sup>nd</sup> smallest i.e b, so string will be  $\overbrace{d}^{\text{a}} \rightarrow$  "ab cda"

\* Approach :- ( $s = ccad$ )

Take a set()  $S = a, c, d$

Now iterate 1 by 1 each char in string.  
 $ccad$ , we don't need to its replace  
 $c$  so remove  $c$  from set,

$S = a, b$  next

Now find the smallest element after  $c$ .  
 i.e.  $a < c$  present after  $c$ .  
 replace  $a$  with  $c$  &  $c$  by  $a$ .

\* Code :-

string chooseup(string a) {

E

set<char> S;

for (int i=0; i < a.length(); i++) S.insert(a[i]);

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

B. erase(a[i]);

if (S.empty()) break;

char ch = \*S.begin();

if (ch < a[i]) {

int ch2 = a[i];

for (int j=0; j < a.length(); j++)

{ if (a[j] == ch) a[j] = ch2;

else if (a[j] == ch2) a[j] = ch;

} // End of loop

break;

3

3

return a;

3

Ques. • Maximum train for which stoppage can be provided

n-platform & 2 main running railway track for both direct. Trains which need to be ~~occupy~~ stop at your station must occupy one platform for their stoppage & the train which need not to stop at your station will run away through either of main track without stopping. Now, each train has 3 values, arrival time, departure time, and platform number. we are given 'm' such trains you have to tell max. no. of train for which you can provide stoppage at your station.

\* Example

$$n = 3, m = 6$$

Train no. Arrival time Dept. time ptt. no.

Train no.	Arrival time	Dept. time	ptt. no.
1	10:00	10:30	1
2	10:10	10:30	1
3	10:00	10:20	2
4	10:30	12:30	2
5	12:00	12:30	3
6	09:00	10:05	1

O/P  $\Rightarrow$  max. stoppage train = 5

If train no. 1 will left to go without stoppage then 2 and 6 can easily be accomplished on platform 1.

3 & 4 on ptt. 2 & 5 on platform 3.

Intuition

- for  $n$  platforms we'll simply make  $n$ -vectors and put respective trains in those vector acc. to platform no.
  - After that by apply greedy approach, we can solve easily.
- NOTE :** we'll take input in form of 4 digit integer for arrival & departure time as 1030 will represent 10:30.
- Also, we will choose a 2-D array for input as  $\text{arr}[m][3]$  where  $\text{arr}[i][0]$  denotes arrival time  $\text{arr}[i][1]$  denotes departure time and  $\text{arr}[i][2]$  denotes the platform for  $i$ th train

## Ques • Minimum Platform problem

Given arrival and departure time of all trains that reach railway station. Find min. no. of platforms required for the railway station so that no train is kept waiting.

Consider all train arrives at same day & leave on same day. Arrival & dept. time can never be same for a train but arrival time of 1 train can equal to dept time of another train. At a given instance of time, same platform can't be used for both dept of train & arrival of another train.

### \* Example :-

$$n = 6$$

$$\text{arr[ }]=\{0900, 0940, 0950, 1100, 1500, 1800\}$$

$$\text{dept[ }]=\{0910, 1200, 1120, 1130, 1900, 2000\}$$

$$\text{output} = 3$$

min. 3 platform are required to safely arrive and depart all trains.

### \* Intuition :-

The idea is to consider all events in sorted order. Once the events are in sorted order, trace the no. of trains at any time keeping track of trains that have arrived, but not departed.

## \* Algorithm

- ① sort arrival and departure times of trains.
- ② Create 2 pointers  $i=0$ , and  $j=0$  and a variable to store ans and current count plat.
- ③ Run a loop, while  $i < n$  and  $j < n$   
 & compare the  $i^{th}$  element of arrival array and  $j^{th}$  element of departure array.
- ④ If arrival time  $<$  dept. time, then  
 1 or more platform is needed so inc  
 count, i.e., plat++ and increment ' $i$ '
- ⑤ else if arrival time  $>$  dept time.  
 then, one less platform is needed so  
 use the count i.e., plat-- and inc  $j$ .
- ⑥ update the ans,  
 i.e.  $ans = \max(ans, plat)$

Ques → Buy max. stock if i stocks can be bought on i-th day

There is product with infinite stocks.

The stock price given for n-days, where  $a[i]$  denotes price of stock on  $i^{th}$  day.  
A rule that, customer can buy at most  $i$  stocks on  $i^{th}$  day. If customer has  $K$  amount of money initially, find out max. no. of stocks customer can buy.

### ④ EXAMPLE

$$\text{price}[] = \{10, 7, 19\} \Rightarrow 4 \text{ ans}$$

$K = 45$ , customer purchase 1 stock on day 1, 2 stock on day 2, 1 stock on day 3,

$$10 + (7+7) + 19 = 43 \Rightarrow 4 \text{ stock.}$$

### ⑤ Intution :-

Greedily, we'll buy the stock whose price is lowest, the 2<sup>nd</sup> smallest lowest price stock, then 3<sup>rd</sup> lowest price or

sort the array, and arrange with pair of index. as:

$$(7, 2), (10, 1), (19, 3)$$

price max. stock.

Now we check for available money as  $14 \leq 45$  (if less than available money)  
update  $K$ , i.e.  $\Rightarrow 36$ .

maintains a variable 'a' as it holds  
no. of stock.

$$a = 2.$$

Now again we check for  $10 \leq (K - \text{updated})$

If yes update 'a'.

$$\text{Now } \Rightarrow a = 2 + 1, K = 31 - 10 \Rightarrow 21.$$

Now again check  $19 \times 3 > 21$ .

$$\frac{\text{divide } K}{\text{Bucl per stock}} \Rightarrow \frac{21}{19} \approx 1$$

$$\therefore a = 2 + 1 + 1$$

return a.

### \* Code :-

```
for (int i=0; i<n; i++) cin >> a[i];
```

```
vector<pair<int, int>> v;
```

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

```
v.push_back({a[i], i+1});
```

```
sort(v.begin(), v.end());
```

```
int ans = 0;
```

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

```
    int price = v[i].first;
```

```
    int stock = v[i].second;
```

```
    if (price * stock <= K) {
```

```
        ans += stock;
```

```
        K -= (price * stock);
```

(3)

else (3)

```
    ans += (K / price);
```

```
    K -= price * (K / price);
```

(3)

cout << ans;

(3)

Ques - Find min. and max. amount to buy all N candies.

In a confectionery store  $N$  different types of candies are available & the price of all the  $N$  different types of candies are provided to you. You are now provided with an attractive offer.

You can buy single candy & get almost  $k$  other candies for free.

Now you've to answer 2 questions, firstly find what is the min. amount of money you can spend to buy all  $N$  diff. candies. 2<sup>nd</sup> find max. amount you've to spend to buy all the  $N$  diff. candies.

Example :-

$$N = 4$$

$$k = 2$$

$$\text{candies}[] = \{3, 2, 1, 4\}$$

$$O/P = 3, 7.$$

Intuition :-

We must use the offer & get max. candies back for every purchase. So if we want to minimize the money, we must buy candies at min. cost and get candies of max. cost for free. To maximize the money, we must do the reverse.

\* for finding min amount :-

Start purchasing from starting and reduce K free candies from last with every single purchase.

\* for finding max. amount :-

Start purchasing candies from the end and reduce K free candies from starting in every single purchase

Code :-

```
(1) int findMin (int arr[], int n, int k)
    [E]     int res = 0;
    for (int i=0; i<n; i++)
        res += arr[i];
        n = n - k;
    [3]     return res;
```

```
int findMax (int arr[], int n, int k)
```

```
[E]     int res=0; index = 0;
    for (int i=n-1; i>=index; i--)
        [3]         res += arr[i];
        index += k;
```

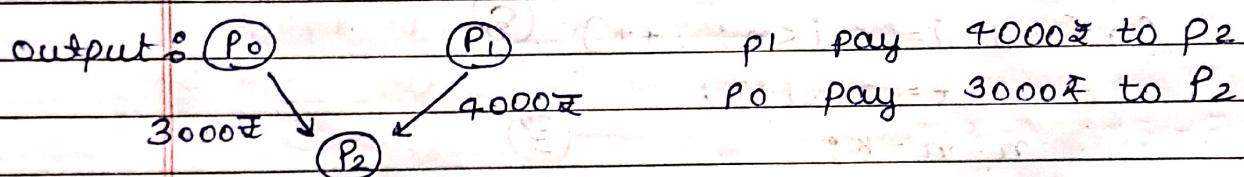
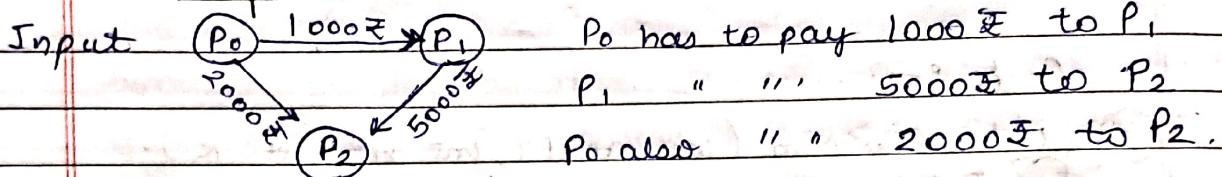
```
return res;
```

```
[3]
```

Ques → Maximize cashflow among given set of friend who borrowed money from each other.

Given no of friends who've to give or take some amount of money from one another design algo. by which total cash flow among all the friends is minimized.

Example



Intuition (greedy approach)

(1) Do following step for every person P<sub>i</sub> where i is from 0 to n-1

(1) Compute net amt for every person. The net amount for person 'i' can be computed by subtracting sum of all debts by sum of all credit.

amount[maxCredit] - = min;

(2) find 2 persons that are max. creditor & debtor. Let max amount to be credited max. creditor be  $\text{maxCredit}$  & and max. amount to be debited from max. debtor be  $\text{maxDebit}$ . Let the max. debtor be  $Pd$  and max. creditor be  $Pc$ .

```
int mxCredit = getMaxAmount();
int mxDebit = getMinAmount();
```

(3) Find minimum of  $\text{maxDebit}$  &  $\text{max Credit}$ . Let min of 2 be  $x$ . Debit ' $x$ ' from  $Pd$  & credit this amount to  $Pc$ .

(4) If  $x = \text{maxCredit}$ , then remove  $Pc$  from person's & recur for remaining  $(n-1)$  persons.

15) If  $x = \text{maxDebit}$ , remove  $Pd$  from set of persons & recur for remaining  $(n-1)$  persons.

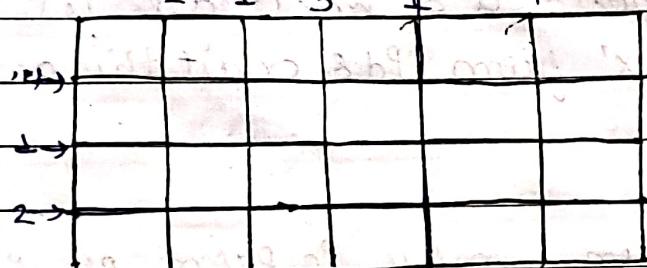
## Ques → Mini cost to cut a board into square

A board of length  $m$  & width  $n$  is given, we need to break this board into  $m \times n$  squares such that cost of breaking is min.

Cutting cost for each edge will be given for the board in short we choose a sequence of cutting so that cost is minimized.

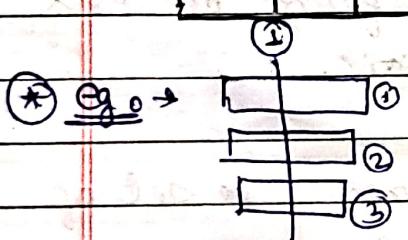
Example

cost of cutting each edge



Init value: total cost = 0

total cost = total cost +  
edge cost \* total pieces



$$\Rightarrow \text{price} = 1, \text{ pieces cut} = 3 \Rightarrow 1 \times 3 = 3$$

cost  $\downarrow$  no. of pieces

when we cut vertically we'll see, how many vertical pieces being cut,

$$\textcircled{*} \quad \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array} \Rightarrow = 2 \times 3 = 6 \quad (6 \text{ will be added in ans})$$

when all cut horizontally, we'll be make a count of horizontal pieces being cut.

## \* How to think?

∴ we think greedy coz we need min. cost  
 so, firstly we cut high cost edges bcoz  
 that time very few no. of pieces available

e.g. - sort in descending order:-

$$(x) = \cancel{4}, 3, 2, 1, \quad (y) = \cancel{4}, 2, 1.$$

we maintain a count var for both horizontal  
 as well as vertical & ans.

$$\begin{array}{|c|} \hline hz = 1 \\ \hline \end{array}; \quad \begin{array}{|c|} \hline vz = 1 \\ \hline \end{array}; \quad \text{ans} = 0;$$

→ Pick from 4 (from x axis) & vz

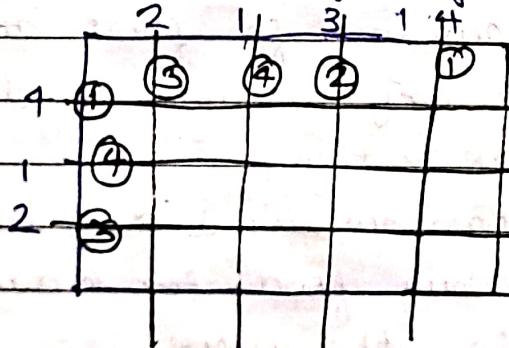
$$\text{ans} = 4 \times 1 \quad (\text{when we cut } hz \text{ by } 1)$$

→ Pick 3 (from x)

$$\text{ans} = 4 \times 1 + 4 \times 2 + 3 \times 2 + 2 \times 2 + 2 \times 4$$

and so on for every cost,

this is a greedy approach to solve



(1, 2, 3, 4 are  
 cutting priority)

Ques → Check if it is possible to survive  
on Island

You are a poor person in an island, there is only 1 shop in an island, open all day of week except Sunday. following const.

N - Max. food you can buy each day

S - No. of days you are required to survive.

M - unit of food reqd. each day to survive.

Right now, its Monday & you need to survive for next 9 days. find min. no. of days you need to buy food from shop. so that you can survive next 9 days.

If not No else say yes.

#### \* EXAMPLE -

I/P  $B = 10, N = 16, M = 2$

O/P - Yes  $\Rightarrow 2$

#### \* Intuition :-

If we can survive for first set 7 days then we can survive any number of days for that we need to check 2 things :-

→ check whether we can survive 1 day or not

→ ( $S >= 7$ ) if we can buy food for 6 days & we can survive for the week i.e.

total food we can buy in a week ( $6 * N$ )

$>=$  total food we required to survive in a week ( $7 * M$ ) then we can survive.

## \* Implementation

```
void surviving (int s, int N, int M)
{
    if ((N * 6) < (M * 7) && s > 6) || M > N)
        cout << "NO";
    else
    {
        int days = (M * s) / N;
        if ((M * s) % N != 0)
            days++;
        cout << "Yes" << days << endl;
    }
}
```

TC - O(1)

SC - O(1)

## Ques - Maximum product subset of an array

Given array 'a', we've to find max. product possible with the subset of elements present in the array. Max. prod. can be single element also.

### \* Example :-

$$a[ ] = \{-1, -1, -2, 4, 3\}$$

$$\text{Output} = 24,$$

$$\text{max product will be } \Rightarrow (-2 * -1 * 4 * 3) = 24$$

### \* Intuition :-

- (1) If there is even no. of -ve no. & no zero, result is simply prod of all.
  - (2) If odd no. of -ve no. & no zero, result is product of all except -ve int. with least abs. value.
  - (3) If there are zeros, then result is product of all except these zeros with 1 exceptional case.
- # The exceptional case is when, 1 -ve no. and all are 0, then result is 0.

Date 9/4/22 Page no: \_\_\_\_\_

Ques → Maximize sum after k-negation

Given an array of int, size N and no. K, you must modifying array arr[] exactly K no. of time. Here modify array means in each operation you can replace any array element either arr[i] by -arr[i] or +arr[i] by arr[i]. You need to perform this operation in such a way that after K operations, sum of array must be max.

② Example :-  
N=5, K=1 arr[] = {1, 2, -3, 4, 5-3}  
Output ⇒ 15  
we've K=1, so that we can change -3 to 3 and sum all element to produce 15 as output.

③ Portion :-  
we just have to replace the min. element arr[i] in array by -arr[i] for current operation.

In this way we can make sum of array maximize after K operation. One interesting case is, once min. element becomes 0, we don't need to make any more changes.

## \* Implementation

```
int MaxSum ( int arr[], int n, int k )
```

**E**

```
    int min = INT_MAX;
```

```
    int max = INT_MIN;
```

```
    int index = -1;
```

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

**E**

```
        if ( arr[j] < min )
```

**E**

```
            min = arr[j];
```

```
            index = j;
```

**E**

```
    if ( min == 0 ) break;
```

```
    arr[index] = -arr[index];
```

```
    int sum = 0;
```

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

```
        sum += arr[i];
```

```
    return sum;
```

**E**

**Ques • Maximize sum( $\text{arr}[i] * i$ ) of array**

Given an array A of N size, task is write a prog. to find max. value of  $\sum \text{arr}[i] * i$ , where  $i = 0, 1, 2, \dots, n-1$ .

**④ Example :-**

$$\text{Arr}[i] = \{5, 3, 2, 4, 1\}$$

$$\text{O/P} = 40$$

If we arrange array as,  $1, 2, 3, 4, 5$  then we can see that min. index will multiply with min. no. & max. index will multiply with max. no.

**⑤ Intuition :-**

The fact that the largest value should be scaled max. and the smallest value should be scaled min.

So we multiply the min. value of  $i$ , with the min. value of  $\text{arr}[i]$ . So, sort the given array in rising order and compute of the sum of array  $\sum \text{arr}[i] * i$ , where  $i = 0$  to  $n-1$ .

Snippet

```

int maxSum(int arr[], int n)
{
    sort(arr, arr+n);
    int sum=0;
    for (int i=0; i<n; i++)
        sum += (arr[i]*i);
    return sum;
}

```

(3)

Ques - Maximize sum of absolute difference of any permutation

Given array, we need to find the max. sum of abs. diff. of any permutation of given array.

\* Example :-

Input : {1, 2, 4, 8} O/P = 18

for given array there are several seq like : {2, 1, 4, 8}, {9, 2, 1, 8} & some more

Now abs. diff. of any seq. will be like for this array sequence {1, 2, 4, 8} the abs. diff. sum is

$$= |1-2| + |2-4| + |4-8| + |8-1| = 14$$

for the given array,

$$|1-8| + |8-2| + |2-4| + |4-1| = 18$$

\* Intuition :- (Greedily)

- ① we'll sort the array
- ② Calc. final seq. by taking 1 smallest element and largest element from the sorted array & make one vector array of this final seq.
- ③ finally, calc. sum of abs. b/w elements of array

\* Snippet :-

```

int MaxSumDiff(int arr, int n)
{
    vector<int> final;
    sort(a, a+n);
    for (int i=0; i<n/2; i++)
    {
        final.push_back(a[i]);
        final.push_back(a[n-i-1]); ③
    }
    if (n/2 != 0)
        final.push_back(a[n/2]);
    int maxi = 0;
    for (int i=0; i<n-1; i++) ④
    {
        maxi = maxi + abs(final[i] - final[i+1]);
        ⑤
    }
    maxi = maxi + abs(final[n-1] - final[0]);
    return maxi;
}

```