



# Closest Numbers

by HackerRank

[Problem](#)[Submissions](#)[Leaderboard](#)[Discussions](#)[Editorial](#) [Tutorial](#)

## All topics

[1 Greedy Technique](#)[2 Sorting](#)[3 Pair](#)

## Greedy Technique

A **greedy algorithm** is an algorithm that follows the problem-solving heuristic of making the *locally optimal* choice at each stage with the hope of finding a *global optimum*.

A very common problem which gives good insight into this is the Job Scheduling Problem.

You have  $n$  jobs numbered  $\{1, 2, 3, \dots, n\}$  and you have the start times  $s_i$  and the end times  $e_i$  for the  $i^{\text{th}}$  job. Which jobs should you choose such that you get the maximal set of non-overlapping jobs?

The correct solution to this problem is to sort all the jobs on the basis of their end times and then keep on selecting the job with the minimal index in this list which does not overlap with currently selected jobs.

Sounds intuitive, but why does it work?

Well, since each job has equal weight, selecting the one which makes way for newer jobs sooner is optimal. Although a more formal argument can be made for a rigorous proof, the intuition behind it is similar.

Now, let's consider another problem. You again have  $n$  jobs. Each job can be completed at any time and takes  $t_i$  time to complete and has a point value of  $p_i$ . But with each second, the point value of the  $i^{\text{th}}$  job decreases by  $d_i$ . If you have to complete all the jobs, what is the maximum points that you can get?

The problem basically asks us for an order of accomplishing the jobs.

Here, doing the job with higher  $d_i$  first makes sense. At the same time, doing the job with lower  $t_i$  also sounds good. So how do we decide?

Assume you have just 2 jobs which, without loss of generality, can be numbered as 1 and 2.

Now, if you do job 1 before job 2, your net score is:

$$S_1 = p_1 + p_2 - d_1 t_1 - d_2 t_2 - d_2 t_1$$

Otherwise,

$$S_2 = p_1 + p_2 - d_1 t_1 - d_2 t_2 - d_1 t_2$$

If  $S_1 \geq S_2$  then:

$$\begin{aligned} d_2 t_1 &\leq d_1 t_2 \\ \text{or, } d_1/t_1 &\geq d_2/t_2 \end{aligned}$$

In other words, it is optimal to do job **1** before job **2** iff  $d_1/t_1 \geq d_2/t_2$ .

Notice that this argument can be applied to  $n$  jobs as a sorting rule. The job with maximum  $d_i/t_i$  value should be done first and so on.

This gives us the optimal ordering and is also in line with our intuition.

### Greedy doesn't always work

Greedy solutions are usually good whenever they exist, because they are usually easy to implement and usually have a fast running time. However, *greedy algorithms don't always work!* By this, we don't mean that the greedy algorithm fails to return the correct answer on all inputs. Instead, we mean that the algorithm fails on *at least one* input.

For example, consider the following problem: You again have  $n$  jobs, and the  $i^{\text{th}}$  job takes  $t_i$  time to complete and has a point value of  $p_i$ . This time, the point values do not decrease over time, and you don't have to finish all jobs. Unfortunately you only have a total of  $T$  time to spend. What is the maximum points you can get?

One greedy algorithm that comes to mind is the following: while there is still time remaining, take the job with the largest point value that can be finished within the remaining time. Intuitively, this can be seen to work in some cases. However, this fails in the following set of jobs:

$i$	$t_i$	$p_i$
1	4	100
2	3	95
3	2	90
4	1	5

Assuming  $T = 5$ , the greedy algorithm mentioned above first takes job  $i = 1$ , then job  $i = 4$ , for a total of **105** points. However, this is not the global optimum, because you can take jobs  $i = 2$  and  $i = 3$  for a total of **185** points, which is much higher.

The next greedy algorithm we can try is to always take the job which takes the shortest amount of time to finish. However, this also fails the set of jobs above (where you only get **95** points).

You can probably try crafting a more sophisticated greedy algorithm than the ones we described, but it probably won't work, i.e. it will probably fail on some input. This is because the problem we described to you is equivalent to the [knapsack problem](#) which currently [has no known efficient algorithm](#)!

Related challenge for **Greedy Technique**

#### Pairs



Success Rate: 72.28% Max Score: 50 Difficulty:

Solve Challenge

## Sorting

[Go to Top](#)

Sorting is a technique to re-order the data in increasing or decreasing order.

Data need not be integers; it can be strings, lists or anything which can have comparable objects.

Each programming language comes with an in-built Sort function.

In C, you can use quick sort as follows

```
void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))
```

here

- \*base is pointer to the first element.
- nitems is the number of items to sort.
- size is the size of each element.
- compar is a compare function which can return -1 for less, 0 for equal and +1 for greater.

Sample Code in C

```
#include <stdio.h>
#include <stdlib.h>

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

int main()
{
    int n;

    printf("Before sorting the list is: \n");
    for( n = 0 ; n < 5; n++ )
    {
        printf("%d ", values[n]);
    }

    qsort(values, 5, sizeof(int), cmpfunc);

    printf("\nAfter sorting the list is: \n");
    for( n = 0 ; n < 5; n++ )
    {
        printf("%d ", values[n]);
    }

    return(0);
}
```

In C++ algorithm library comes with built in sort function.

example using array and a vector

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a[] = {5,4,3,2,1};
    vector<int> b(a,a+5); //initialised b using elements of a.
    sort(a,a+5);
    sort(b.begin(),b.end());
    for (auto i :a) {
        cout<<i<<" ";
    }
    cout<<endl;
    for (auto i:b) {
        cout<<i<<" ";
    }
    cout<<endl;
}
```

Note that you can also give your custom compare function in the third argument.  
In order to reverse sort you can do

```
sort(b.rbegin(),b.rend());
```

In Python you can sort using

```
a.sort()
```

sometimes you can even define key which is like compare function.

```
arr = [[3, 4, 2, 1], [2, 1, 3, 4]]
arr.sort(key = lambda x : x[1]) #key should be a callable function
#here lambda is a returns the 1st element inside the list.
#arr is now [[2, 1, 3, 4], [3, 4, 2, 1]], sorted on 1st index.
```

Sample Code in Java :

```
import java.util.* ;
public class sorting{
    public static void main(String []args){
        Scanner sc = new Scanner(System.in) ;
        int n ;
        n = sc.nextInt() ;
        int A[] = new int[n] ;
        for(int i=0;i<n;i++)
            A[i] = sc.nextInt() ;
        Arrays.sort(A) ;
        for(int i=0;i<n;i++)
            System.out.print(A[i] + " " ) ;
        System.out.print("\n") ;
    }
}
```

Related challenge for **Sorting**

**Pairs**



Success Rate: 72.28% Max Score: 50 Difficulty:

Solve Challenge

## Pair

A pair as given by

```
pair <type_a, type_b> p;
//Here type_a, type_b can be anything int, string , char etc.
```

A lot of things can be easily stored using a pair.

- X,Y coordinates.
- Complex numbers
- index,value
- Fraction numbers

A very interesting use is to perform a sort.

### Example:

Suppose you are given a data with values and you want to return the index of the  $n^{th}$  smallest element

You can store pairs of (value, index) in a vector and sort it according to the values. You get sorted values and an arbitrary order index. Also, for two equal values lower index comes before higher index.

Now returning index of  $k^{th}$  sorted element is simply `arr[k].second`.

### Note:

- User can define his own comparator function for sorting vector of pairs.
- It can nested like `pair< int,int>,int>`.

Sample C ++ code :

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

bool compare(const pair<int,int> &a,const pair<int,int> &b){
    if(a.first < b.first)
        return 1 ;
    else if(a.first == b.first){
        if( a.second < b.second )
            return 1 ;
    }
    return 0 ;
}

int main(){
    int n ;
    cin >> n ; // number of elements
    vector<pair<int,int> > A(n) ;
    for(int i=0;i<n;i++){
        int x ; cin >> x ;
        A[i].first = x ;
        A[i].second = i ;
    }
    sort(A.begin(),A.end()) ; // sorting according to default function
    sort(A.begin(),A.end(),compare) ; // exactly same as above ( default function is overloaded )
    return 0 ;
}
```

---

Join us on IRC at [#hackerrank](#) on freenode for hugs or bugs.

[Contest Calendar](#) | [Interview Prep](#) | [Blog](#) | [Scoring](#) | [Environment](#) | [FAQ](#) | [About Us](#) | [Support](#) | [Careers](#) | [Terms Of Service](#) | [Privacy Policy](#) | [Request a Feature](#)