University of Information Technology and Sciences (UITS)

Lab Report - 3

IT-214 : Algorithm Lab

Sorting and greedy algorithm

Submitted To:

Sumaiya Akhtar Mitu
Lecturer,
Department of IT, UITS

Submitted By:

Name: Nazmul Zaman
Student ID:2014755055
Department of IT UITS

24 MAY 2022

Contents

1 Abstrac

In mathematics and computer science, an algorithm is a finite sequence of well-defined instructions, typically used to solve a class of specific problems or to perform a computation. Algorithms are used as specifications for performing calculations and data processing.

2 Introduction

In this task we are going to learn about sorting and greedy algorithm and there different methods.

Sorting algorithm : A sorting algorithm is a method for reorganizing a large number of items into a specific order, such as alphabetical, highest-to-lowest value or shortest-to-longest distance. Sorting algorithms take lists of items as input data, perform specific operations on those lists and deliver ordered arrays as output.

Greedy algorithm : Image result for what is greedy algorithm A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result. The algorithm never reverses the earlier decision even if the choice is wrong.

1. Quick Sort : QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways

2. Merge Sort : Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then it merges the two sorted halves

3. Fractional Knapsack : The fractional knapsack problem is also one of the techniques which are used to solve the knapsack problem.

4. Job Scheduling with deadline : A Simple Solution is to generate all subsets of a given set of jobs and check individual subsets for the feasibility of jobs in that subset.

3 Working methods

1.Quick Sort :
Step 1 − Choose the highest index value has pivot
Step 2 − Take two variables to point left and right of the list excluding pivot
Step 3 − left points to the low index
Step 4 − right points to the high
Step 5 − while value at left is less than pivot move right
Step 6 − while value at right is greater than pivot move left
Step 7 − if both step 5 and step 6 does not match swap left and right
Step 8 − if left $\geq$ right, the point where they met is new pivot

The time complexity of quicksort is O(n*logn).

2.Merge Sort :
Step 1 − if it is only one element in the list it is already sorted, return.
Step 2 − divide the list recursively into two halves until it can no more be divided.
Step 3 − merge the smaller lists into new list in sorted order.
The time complexity of MergeSort is O(n*Log n)

3. Fractional Knapsack :
The time complexity of the fractional knapsack problem is O(NlogN).

Job Scheduling with deadline :
1) Sort all jobs in decreasing order of profit.
2) Iterate on jobs in decreasing order of profit.For each job , do the following :
a)Find a time slot i, such that slot is empty and i ¡ deadline and i is greatest.Put the job in this slot and mark this slot filled.
b)If no such i exists, then ignore the job.

Job sequencing problems has the time complexity of O(n2).

4 Source Code

4.1 (Merge Sort)

```
1
2  // NAZMUL ZAMAN BSC IN IT
3
4  #include<bits/stdc++.h>
5  using namespace std;
6
7
8  void merge(int arr[],int l,int mid ,int r )
9  {
10     int n1=mid-l+1;
11     int n2=r-mid;
12     int a[n1];
13     int b[n2];
14
15     for(int i=0;i<n1;i++)
16     {
17         a[i]=arr[l+i];
18
19     }
20
21     for(int i=0;i<n2;i++)
22     {
23         b[i]=arr[mid +1+i];
24     }
25
26     int i=0;
27     int j=0;
28     int k=l;
29
30     while(i<n1 && j<n2)
31     {
32         if(a[i]<b[j])
33         {
34             arr[k]=a[i];
35             k++,i++;
36         }
37         else
38         {
39             arr[k]=b[j];
40             k++,j++;
41
42         }
43     }
44     while(i<n1)
45     {
46            arr[k]=a[i];
47             k++,i++;
```

```
48      }
49      while(i<n2)
50      {
51            arr[k]=b[j];
52               k++,j++;
53      }
54
55  }
56  /* Function to print an array */
57  void printArray(int arr[], int n)
58
59  {
60    int i;
61    for (i = 0; i < n; i++)
62      cout << arr[i] << " ";
63    cout << endl;
64  }
65
66  void mergeSort(int arr[],int l,int r)
67  {
68       if(l<r) //if l==r then  there will no array for divide
69       {
70            int mid=(l+r)/2;
71            mergeSort(arr,l,mid);
72            mergeSort(arr,mid+1,r); // till this function we sorted
      both array
73
74            merge(arr,l,mid,r); // this function for merger the sorted
      array
75
76       }
77  }
78
79
80
81  int main()
82  {
83        cout<<"Enter the size of an array : ";
84    int n;
85       cin>>n;
86       int arr[n];
87       for(int i=0;i<n;i++)
88       {
89            cout<<"Enter the value of an array "<< i <<" index is  : ";
90       cin>>arr[i];
91
92       }
93       cout<<"\nbefore sorted the array : ";
94        for(int i=0;i<n;i++)
95       {
96       cout<<arr[i]<<" ";
```

```
97
98        }
99          mergeSort(arr, 0, n - 1);
100     cout << "\nSorted array: \n";
101     printArray(arr, n);
102     return 0;
103 }
```

OUTPUT ?

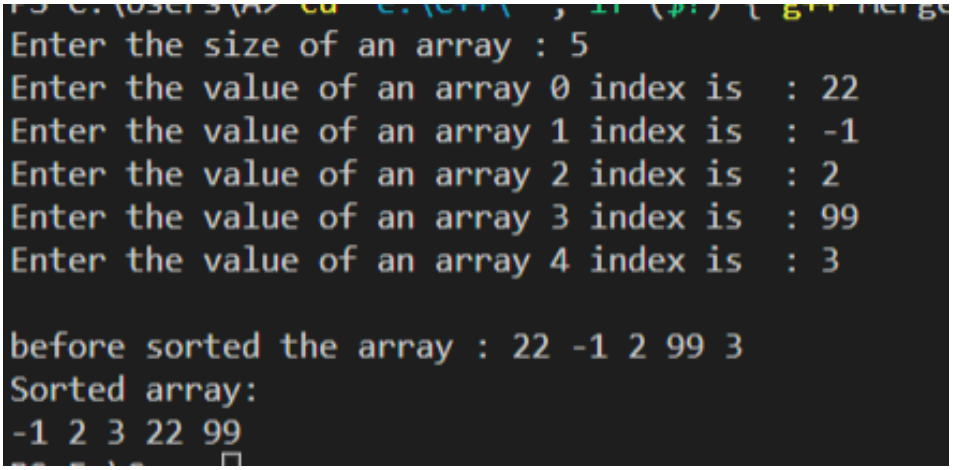

Figure 1: OUTPUT

4.2 (Quick Sort)

```
// NAZMUL ZAMAN BSC IN IT

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


// A utility function to swap two elements
void swap(int* a, int* b)
{
  int t = *a;
  *a = *b;
  *b = t;
}




/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition (int arr[], int low, int high)
{
  int pivot = arr[high]; // pivot
  int i = (low - 1); // Index of smaller element and indicates the
   right position of pivot found so far

  for (int j = low; j <= high - 1; j++)
  {
    // If current element is smaller than the pivot
    if (arr[j] < pivot)
    {
      i++; // increment index of smaller element
      swap(&arr[i], &arr[j]);
    }
  }
  swap(&arr[i + 1], &arr[high]);
  return (i + 1);
}



/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
  if (low < high)
```

```
50    {
51      /* pi is partitioning index, arr[p] is now
52      at right place */
53      int pi = partition(arr, low, high);
54
55      // Separately sort elements before
56      // partition and after partition
57      quickSort(arr, low, pi - 1);
58      quickSort(arr, pi + 1, high);
59    }
60  }
61
62
63
64  /* Function to print an array */
65  void printArray(int arr[], int size)
66
67  {
68    int i;
69    for (i = 0; i < size; i++)
70      cout << arr[i] << " ";
71    cout << endl;
72  }
73
74
75
76  int main()
77  {
78      cout<<"Enter the size of an array : ";
79    int n;
80      cin>>n;
81      int arr[n];
82      for(int i=0;i<n;i++)
83      {
84          cout<<"Enter the value of an array "<< i <<" index is  : ";
85      cin>>arr[i];
86
87      }
88      cout<<"\nbefore sorted the array : ";
89       for(int i=0;i<n;i++)
90      {
91      cout<<arr[i]<<" ";
92
93      }
94    quickSort(arr, 0, n - 1);
95    cout << "\nSorted array: \n";
96    printArray(arr, n);
97    return 0;
98  }
```

?

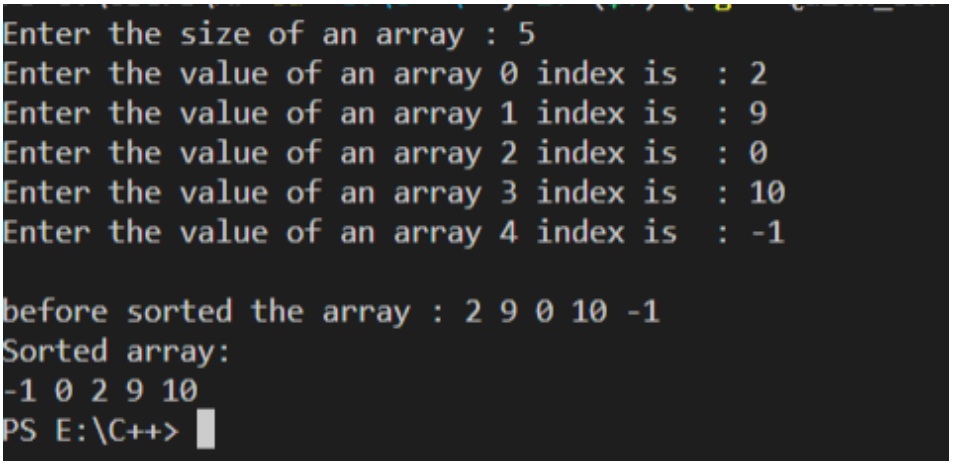

Figure 2: OUTPUT

4.3 (Job Sequence)

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

// A structure to represent a job
struct Job
{
char id;
int dead;
int profit;
};

// This function is used for sorting all the jobs according to the
   profit
bool compare(Job a, Job b)
{
  return (a.profit > b.profit);
}


void jobschedule(Job arr[], int n)
{
  // Sort all jobs according to decreasing order of prfit
  sort(arr, arr+n, compare);

  int result[n]; // To store result
  bool slot[n];

  // Initialize all slots to be free
  for (int i=0; i<n; i++)
    slot[i] = false;


  for (int i=0; i<n; i++)
  {
  // Find a free slot for this job (Note that we start
  // from the last possible slot)
  for (int j=min(n, arr[i].dead)-1; j>=0; j--)
  {
    // Free slot found
    if (slot[j]==false)
    {
      result[j] = i; // Add this job to result
      slot[j] = true; // Make this slot occupied
      break;
    }
  }
  }

  // Print the result
```

```
50    for (int i=0; i<n; i++)
51    if (slot[i])
52      cout << arr[result[i]].id << " ";
53  }
54
55
56  int main()
57  {
58    Job arr[] = { {'a', 2, 20}, {'b', 2, 15}, {'c', 1, 10},
59          {'d', 3, 5}, {'e', 3, 1}};
60    int n = 5;
61    cout << "maximum profit sequence of jobs is-->";
62    jobschedule(arr, n);
63
64  }
```

?

```
PS C:\Users\A> cd "e:\C++\" ; if ($?) { g++ job_sequence.cpp -o job_sequence } ; if ($?) { .\job_sequence }
maximum profit sequence of jobs is-->b a d
PS E:\C++>
```

Figure 3: OUTPUT

4.4 (Fractional Knapsack)

```cpp
// NAZMUL ZAMAN BSC IN IT

#include <bits/stdc++.h>

using namespace std;

// Structure for an item which stores weight and
// corresponding value of Item
struct Item {
  int value, weight;

  // Constructor
  Item(int value, int weight)
  {
    this->value = value;
    this->weight = weight;
  }
};

// Comparison function to sort Item according to val/weight
// ratio
bool cmp(struct Item a, struct Item b)
{
  double r1 = (double)a.value / (double)a.weight;
  double r2 = (double)b.value / (double)b.weight;
  return r1 > r2;
}

// Main greedy function to solve problem
double fractionalKnapsack(int W, struct Item arr[], int n)
{
  // sorting Item on basis of ratio
  sort(arr, arr + n, cmp);

  // Uncomment to see new order of Items with their
  // ratio
  /*
  for (int i = 0; i < n; i++)
  {
    cout << arr[i].value << " " << arr[i].weight << " :
  "
       << ((double)arr[i].value / arr[i].weight) <<
  endl;
  }
  */

  double finalvalue = 0.0; // Result (value in Knapsack)

  // Looping through all Items
  for (int i = 0; i < n; i++) {
```

```
    // If adding Item won't overflow, add it completely
    if (arr[i].weight <= W) {
      W -= arr[i].weight;
      finalvalue += arr[i].value;
    }

    // If we can't add current Item, add fractional part
    // of it
    else {
      finalvalue += arr[i].value
            * ((double)W
              / (double)arr[i].weight);
      break;
    }
  }

  // Returning final value
  return finalvalue;
}

// Driver code
int main()
{
  int W = 50; // Weight of knapsack
  Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };

  int n = sizeof(arr) / sizeof(arr[0]);

  // Function call
  cout << "Maximum value we can obtain = "
    << fractionalKnapsack(W, arr, n);
  return 0;
}
```

?

```
PS C:\Users\A> cd "e:\C++\" ; if ($?) { g++ fractional_Knapsack.cpp -o fractional_Knapsack } ; if ($?) { .\fractional_Knapsack }
Maximum value we can obtain = 240
PS E:\C++>
```

Figure 4: Salary column added in the table

5 Conclusion

In this lab report we learn about different type of sorting algorithm and greedy algorithm and use of those algorithm . A sorting algorithm will put items in a list into an order, such as alphabetical or numerical order and Greedy algorithms are simple instinctive algorithms used for optimization (either maximized or minimized) problems. By using these algorithm we can solve different type of sorting algorithm and greedy alogrithm problem and and it's also helpful for competitive programming.

6 References

1.https://www.w3schools.com/algorithm/algo$_i$*ntro.asp*

2.https://www.javatpoint.com/algo$_i$*ntro.asp*

3.https://github.com/ZamanNazmul