GREEDY APPROACH

Activity Selection Problem using Greedy Approach

You are given n activities with their start and finish times. 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 time.

Example:

```
Example 1 : Consider the following 3 activities sorted by
by finish time.
     start[]  =  {10, 12, 20};
     finish[] =  {20, 25, 30};
A person can perform at most two activities. The
maximum set of activities that can be executed
is {0, 2} [ These are indexes in start[] and
finish[] ]

Example 2 : Consider the following 6 activities
sorted by by finish time.
     start[]  =  {1, 3, 0, 5, 8, 5};
     finish[] =  {2, 4, 6, 7, 9, 9};
A person can perform at most four activities. The
maximum set of activities that can be executed
is {0, 1, 3, 4} [ These are indexes in start[] and
finish[] ]
```

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

1) Sort the activities according to their finishing time

2) Select the first activity from the sorted array and print it.

3) Do following for remaining activities in the sorted array.

…….a) If the start time of this activity is greater than or equal to the finish time of previously selected activity then select this activity and print it.

In the following C implementation, it is assumed that the activities are already sorted according to their finish time.

```
// C++ program for activity selection problem.
// The following implementation assumes that the activities
// are already sorted according to their finish time
#include<stdio.h>

// Prints a maximum set of activities that can be done by a single
// person, one at a time.
//  n   -->  Total number of activities
//  s[] -->  An array that contains start time of all activities
//  f[] -->  An array that contains finish time of all activities
void printMaxActivities(int s[], int f[], int n)
{
    int i, j;

    printf ("Following activities are selected n");

    // The first activity always gets selected
    i = 0;
    printf("%d ", i);

    // Consider rest of the activities
    for (j = 1; j < n; j++)
    {
      // If this activity has start time greater than or
      // equal to the finish time of previously selected
      // activity, then select it
```

```
       if (s[j] >= f[i])
       {
           printf ("%d ", j);
           i = j;
       }
     }
}

// driver program to test above function
int main()
{
    int s[] =  {1, 3, 0, 5, 8, 5};
    int f[] =  {2, 4, 6, 7, 9, 9};
    int n = sizeof(s)/sizeof(s[0]);
    printMaxActivities(s, f, n);
    return 0;
}
```

Output:

```
Following activities are selected
0 1 3 4
```

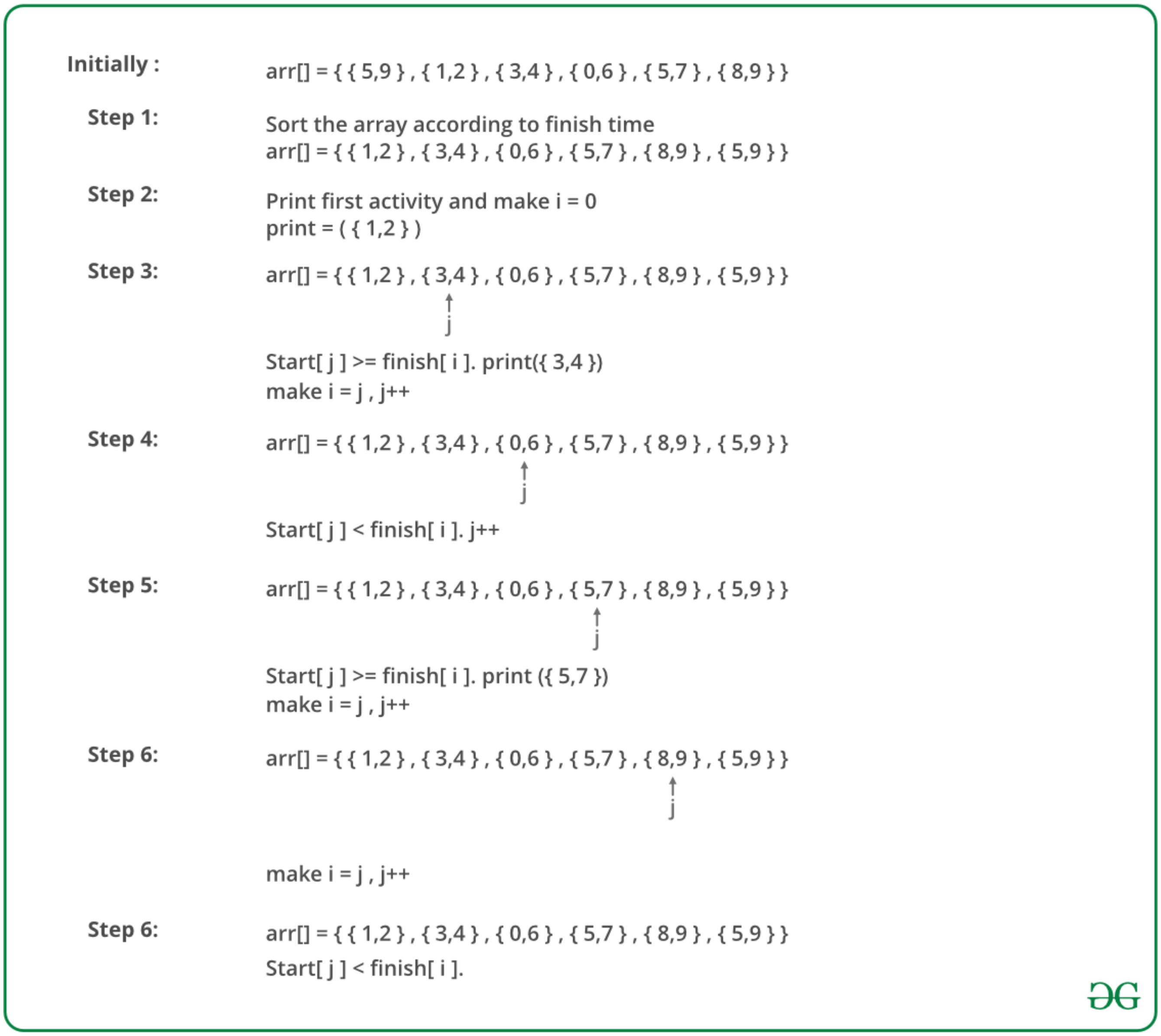
How does Greedy Choice work for Activities sorted according to finish time?

Let the given set of activities be S = {1, 2, 3, ..n} and activities be sorted by finish time. The greedy choice is to always pick activity 1. How come the activity 1 always provides one of the optimal solutions. We can prove it by showing that if there is another solution B with the first activity other than 1, then there is also a solution A of the same size with activity 1 as the first activity. Let the first activity selected by B be k, then there always exist A = {B – {k}} U {1}.(Note that the activities in B are independent and k has smallest finishing time among all. Since k is not 1, finish(k) >= finish(1)).

How to implement when given activities are not sorted?

We create a structure/class for activities. We sort all activities by finish time. Once we have activities sorted, we apply same above algorithm.

Below image is an illustration of the above approach:



Initially : arr[] = { { 5,9 } , { 1,2 } , { 3,4 } , { 0,6 } , { 5,7 } , { 8,9 } }

Step 1: Sort the array according to finish time
arr[] = { { 1,2 } , { 3,4 } , { 0,6 } , { 5,7 } , { 8,9 } , { 5,9 } }

Step 2: Print first activity and make i = 0
print = ({ 1,2 })

Step 3: arr[] = { { 1,2 } , { 3,4 } , { 0,6 } , { 5,7 } , { 8,9 } , { 5,9 } }
j
Start[j] >= finish[i]. print({ 3,4 })
make i = j , j++

Step 4: arr[] = { { 1,2 } , { 3,4 } , { 0,6 } , { 5,7 } , { 8,9 } , { 5,9 } }
j
Start[j] < finish[i]. j++

Step 5: arr[] = { { 1,2 } , { 3,4 } , { 0,6 } , { 5,7 } , { 8,9 } , { 5,9 } }
j
Start[j] >= finish[i]. print ({ 5,7 })
make i = j , j++

Step 6: arr[] = { { 1,2 } , { 3,4 } , { 0,6 } , { 5,7 } , { 8,9 } , { 5,9 } }
j
make i = j , j++

Step 6: arr[] = { { 1,2 } , { 3,4 } , { 0,6 } , { 5,7 } , { 8,9 } , { 5,9 } }
Start[j] < finish[i].