

Activity Selection problem is a approach of selecting non-conflicting tasks based on start and end time and can be solved in $O(N \log N)$ time using a simple greedy approach. Modifications of this problem are complex and interesting which we will explore as well. Suprising, if we use a Dynamic Programming approach, the time complexity will be $O(N^3)$ that is lower performance.

```
Greedy-Iterative-Activity-Selector(A, s, f):
```

```
    Sort A by finish times stored in f
```

```
    S = {A[1]}
```

```
    k = 1
```

```
    n = A.length
```

```
    for i = 2 to n:
```

```
        if s[i] ≥ f[k]:
```

```
            S = S U {A[i]}
```

```
            k = i
```

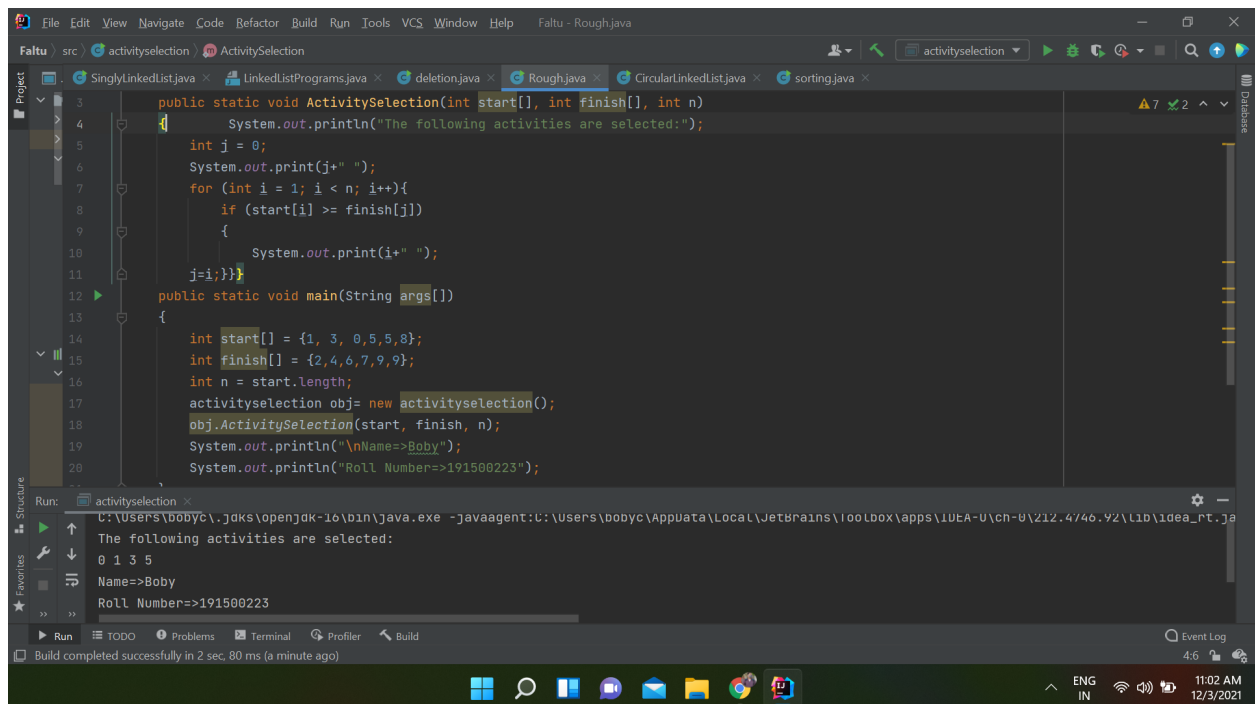
```
    return S
```

Complexity

Time Complexity:

When activities are sorted by their finish time: $O(N)$

When activities are not sorted by their finish time, the time complexity is $O(N \log N)$ due to complexity of sorting



The screenshot shows an IDE with the following Java code in `ActivitySelection.java`:

```
public static void ActivitySelection(int start[], int finish[], int n)
{
    System.out.println("The following activities are selected:");
    int j = 0;
    System.out.print(j+" ");
    for (int i = 1; i < n; i++){
        if (start[i] >= finish[j])
        {
            System.out.print(i+" ");
            j=i;
        }
    }
}

public static void main(String args[])
{
    int start[] = {1, 3, 0, 5, 8};
    int finish[] = {2, 4, 6, 7, 9};
    int n = start.length;
    activityselection obj= new activityselection();
    obj.ActivitySelection(start, finish, n);
    System.out.println("\nName=>Bobby");
    System.out.println("Roll Number=>191500223");
}
```

The Run window shows the following output:

```
The following activities are selected:
0 1 3 5
Name=>Bobby
Roll Number=>191500223
```

The status bar at the bottom indicates: Build completed successfully in 2 sec, 80 ms (a minute ago).