ESC101 Lecture
25th March 2008

Sujith Thomas
(Y7111037)
sujith@iitk.ac.in

March 26, 2008

Abstract

In the next two lectures the topic of discussion will be problem solving in an efficient manner.

1 Algorithm for sorting

Let us assume that we have an array of some datatype (or object) and we want to sort the array in ascending order. In order to be able to sort the array, any two elements in the array should be comparable i.e. we should be able to say whether an element is greater than, less than or equal to another element.

1.0.1 Selection sort

The intuitive idea behind Selection sort is the following. Suppose we have an array of size n which we need to sort in ascending order. We scan through the array $(n-1)$ times. In i^{th} iteration we find the i^{th} smallest element. Let i^{th} smallest element be in position t. We then swap the elements at index i with the element at index t. At the end of $(n-1)^{th}$ iteration we will get an array sorted in the ascending order.

The following function will give us the i^{th} smallest element. Here the assumption is that i^{th} element is the smallest element in the index range i to n.

```
static int smallestNum(int a[],int i) {
        int t = i; // t stores the index of smallest element
        if(a==null)
                throw newNullPointerEx; // Throw exception if
                        // array a is null
        for(int j=t+1;j<a.length;j++)
                if(a[t] > a[j]) t=j;
        return t;
}
```

The method below uses the method smallestNum to sort an array a.

```
static void selectionSort(int a[]) {
        int i;
        if(a==null) return;
        for(i=0;i<a.length - 1;i++) {
                int t = smallestNum(a,i);
                int x = a[t];
                a[t] = a[i];
                a[i] = x;
        }
}
```

1.0.2 Time and Space Complexity

We need to bring in a measure of execution time for the above algorithm. We call this time complexity of the algorithm and we express it in terms of the size of the input. In the selection sort algorithm above the time complexity will be defined in terms of the number of elements in array a. Let us assume that every statement takes constant time c. We can make this assumption because we are only trying to find the relation between the input size and the execution time. Then the time taken by each line in the function smallestNum will be as follows.

```
static int smallestNum(int a[],int i) {
        int t;
        t=i; // time c
        if(a==null) throw newNullPointerEx; // time c
        for(int j=i+1;j<a.length;j++) // time c executed (n-i-1) times
        if(a[t] > a[i]) t=j; // time c executed (n-i-1) times
        return t; // time c
}
```

Therefore the time taken by smallestNum method will be
$(3c + 2c(n - i - 1))$. Similarly time taken by each line in the function selectionSort will be as follows.

```
static void selectionSort(int a[]) {
        int i;
        if(a==null) return;
        for(i=0;i < a.length - 1;i++) { // time c executed (n-1) times
                int t = smallestNum(a,i); // time (3c+2c(n-i-1))
                int x = a[t]; // time c
                a[t] = a[i]; // time c
                a[i] = x; // time c
        }
}
```

The time taken by selectionSort will be $\Sigma_{i=0}^{n-1}\{6c + 2c(n - i - 1)\}$
$= K_0 + K_1(n^2 - n)$.

We can ignore K_0 and K_1 because they are dependent on machine. The value of $(n^2 - n)$ is almost same as n^2 for large values of n. This is called the asymptotic behaviour of the algorithm. Therefore we say the time complexity of the above algorithm is 'order of n-squared' denoted by $O(n^2)$. This is called the Big-oh notation.

The space complexity of the above Selection sort algorithm is $O(1)$ which means space required by the algorithm is constant. Intuitively this means that the space required by the algorithm is not a function of the input size.

Suppose space complexity of two algorithms is $O(n^3)$ and $O(n^2 log(n))$. For large values of n $O(n^2 log(n))$ algorithm will be better than $O(n^3)$ algorithm.

Therefore if the space and time complexities of algorithms are given it helps us to choose the better algorithm.