ECE374 Fall2020

Lab6: Randomized Selection

Name: Zhang Yichi 3180111309

Nov. 10^{th} 2020

1 Introduction

In this lab, we will implement and analyze the randomized selection algorithm.

2 Python Code for Randomized Selection

```
def randomized_partition(A,p,r):
    rand = np.random.randint(p,r)
    A[r],A[rand] = A[rand],A[r]
    x = A[r]
    i = p-1
    for j in range(p,r):
        if A[j] <= x:
            i += 1
            A[i],A[j] = A[j],A[i]
    A[i+1],A[r] = A[r],A[i+1]
    return i+1

def randomized_select(A,p,r,i):
    if p == r:
        return A[p]
    q = randomized_partition(A,p,r)
    k = q - p + 1
    if i == k:
        return A[q]
    elif i < k:
        return randomized_select(A,p,q-1,i)
    else:
        return randomized_select(A,q+1,r,i-k)
```

This algorithm uses recursion to find the i^{th} largest number in the array A. The principle here is similar to the randomized quick sort. It uses randomized partition to first pick an element and place all the elements smaller before it and all the elements larger after it. In this way, we know this element is the j^{th} largest number in this array if there are the return value is j. To find the i^{th} largest number, we recursively do the randomized partition until $j = i$.

3 Test Example for Randomized Selection

```
A = np.random.randint(0,100,37)
print(A)
min = randomized_select(A,0,len(A)-1,1)
max = randomized_select(A,0,len(A)-1,len(A))
median = randomized_select(A,0,len(A)-1,(len(A)+1)//2)
print(f"min is {min}, max is {max}, median is {median}")
print(sorted(A))
```

```
[12 28 79 60 14 91 81 19 13 92 78 86 45 30 17 81 34 14 20 41 83 13 71 74
 89 22 37 94 66 27  7 17 40 33 69 77 56]
min is 7, max is 94, median is 41
[7, 12, 13, 13, 14, 14, 17, 17, 19, 20, 22, 27, 28, 30, 33, 34, 37, 40, 41, 45, 56, 60, 66, 69, 71, 74, 77, 78, 79, 81, 81, 83,
 86, 89, 91, 92, 94]
```

Figure 1: randomized selection test result

4 Time Complexity Analysis

Assume the time it takes is T(n) and $T(n) \leq cn$ for sufficiently large c.

$$
\begin{aligned}
T(n) &\leq \frac{1}{n}\sum_{k=0}^{n-1} T(max(k, n-k-1)) + \Theta(n) \text{ (max(k,n-k-1) means falls in larger side of partition)} \\
&\leq \frac{2}{n}\sum_{k=\frac{n}{2}}^{n-1} T(k) + \Theta(n) \text{ (before } \frac{2}{n}\text{, n-k-1 is larger and after } \frac{2}{n} \text{ k is larger and they are symmetric)} \\
&\leq \frac{2}{n}\sum_{k=\frac{n}{2}}^{n-1} ck + \Theta(n) \text{ (assume } T(n) \leq cn \text{ for sufficiently large c)} \\
&= \frac{2c}{n}(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\frac{n}{2}-1} k) + \Theta(n) \\
&= \frac{2c}{n}(\frac{1}{2}(n-1)n - \frac{1}{2}(\frac{n}{2}-1)\frac{n}{2}) + \Theta(n) \\
&= c(n-1) - \frac{c}{2}(\frac{n}{2}-1) + \Theta(n) \\
&= cn - (\frac{cn}{4} + \frac{c}{2} - \Theta(n)) \\
&\leq cn \text{ (this is what we assume and set out to prove)}
\end{aligned}
$$

Therefore, the average time complexity of this algorithm is O(n).