Homework #2
CS 217 @ SJTU
Prof. **Dominik Scheder**
TA. **Tang Shuyang**

Coffee Automaton
ACM Class, Zhiyuan College, SJTU
Due Date: October 7, 2019
Submit Date: October 9, 2019

Exercise 1

If we want to find the maximum item in n items, to ensure it is the maximum, it should be compared with all others directly or indirectly, so any algorithms should be at least $n-1$ comparisons.

If not, there is no comparison between the "maximum" and some others, and we can not ensure its correctness.

Exercise 2

Record the maximum and the minimum.

Fetch 2 items each time, compare them, compare the smaller item and the minimum, and compare the bigger item and the maximum.

The algorithm shows below.

```python
def exercise_2(a, n):  # a is the array, and n is the size
    if n == 1:
        return [a[0], a[0]]
    elif a[0] < a[1]:  # 1 comparison
        max = a[1]
        min = a[0]
    else:
        max = a[0]
        min = a[1]
    for i in range(1, n // 2):  # 3 comparisons for each cycle
        if a[2 * i] < a[2 * i + 1]:
            l = a[2 * i]
            r = a[2 * i + 1]
        else:
            l = a[2 * i + 1]
            r = a[2 * i]
        if l < min:
            min = l
        if r > max:
            max = r

    return [max, min]
```

The algorithm all use $1 + (\frac{n-2}{2}) \times 3 = \frac{3n}{2} - 2$ comparisons.

Exercise 3

Each time compare two items, take the larger one to the next round, and after $k-1$ rounds, there are two items. Compare them, and get the "max" and the "second".

Then find the items those were compared with the "max", compare them with "second". the maximum is the second largest item.

The algorithm shows below.

```
import numpy as np

def exercise_3(a, n, k):  # a is the array, and n is the size n = 2^k
    l = np.zeros(n, k)
    b = np.zeros(n)
    for i in range(0, n):
        b[i] = i
    m = n // 2
    for i in range(0, k - 1):  # 2^(k-i-1) comparisons
        for j in range(0, m):
            if a[b[j]] < a[b[j + 1]]:
                l[b[j + 1]].append(b[j])
                b.pop(j)
            else:
                l[b[j]].append(b[j + 1])
                b.pop(j + 1)
    m = m // 2
    if a[b[0]] < a[b[1]]:  # 1 comparison
        max = a[b[1]]
        second = a[b[0]]
        compare = l[b[1]]
    else:
        max = a[b[0]]
        second = a[b[1]]
        compare = l[b[0]]
    for i in range(0, k):  # k comparisons
        if a[compare[i]] > second:
            second = a[compare[i]]
```

It all uses $\frac{n}{2}+\frac{n}{4}+\cdots+2+1+(k-1)=n+k-2=n+\log_2(n)-2$ comparisons.

Exercise 4

Set the expected number of comparisons as S.

$$\begin{aligned} S &= \mathbf{E}[\sum_{i\neq j} A_{i,j}] \\ &= \sum_{i\neq j} \mathbf{E}[A_{i,j}] \\ &= \sum_i \sum_j \frac{1}{|i-j|+1} - n \end{aligned}$$

Now considering the summation,as matrix (1) shows, the summation of the matrix equals the summation.

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & 1 & \frac{1}{2} & \cdots & \frac{1}{n-1} \\ \cdots & & \cdots & & \\ \frac{1}{n-1} & \frac{1}{n-2} & & \cdots\ 1 & \frac{1}{2} \\ \frac{1}{n} & 1 & \frac{1}{n-1} & \cdots\ \frac{1}{2} & 1 \end{pmatrix}$$

Using the Harmonic number $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$

$$\begin{aligned} S &= \sum_i \sum_j \frac{1}{|i-j|+1} - n \\ &= n \times 1 + 2 \times (n-1) \times \frac{1}{2} + 2 \times (n-2) \times \frac{1}{3} + \cdots + 2 \times (n-(n-1)) \times \frac{1}{n} - n \\ &= 2\left(\frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n} - \frac{1}{2} - \frac{2}{3} - \cdots - \frac{n-1}{n}\right) \\ &= 2\left(n\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) - \left(\frac{1}{2} + \frac{2}{3} + \cdots + \frac{n-1}{n}\right)\right) \\ &= 2\left(n\left(H_n - 1\right) - \left((1-1) + \left(1 - \frac{1}{2}\right) + \left(1 - \frac{1}{3}\right) + \cdots + \left(1 - \frac{1}{n}\right)\right)\right) \\ &= 2\left(n\left(H_n - 1\right) - \left(n - H_n\right)\right) \\ &= 2\left(n+1\right) H_n - 4n \end{aligned}$$

Which is the expected number of comparisons.

Exercise 5

The QuickSelect method first chooses a random pivot k, and divide the array into k and two parts: bigger than k, smaller than k, which is same as quicksort.

However, quickselect only concern one of the two parts, and omit the other, while quicksort concern the both. So we can say that quickselect is a "partial execution" of quicksort.

The example quicksort tree to find 4 shows in Figure 1, and the red part is visited by quickselect.

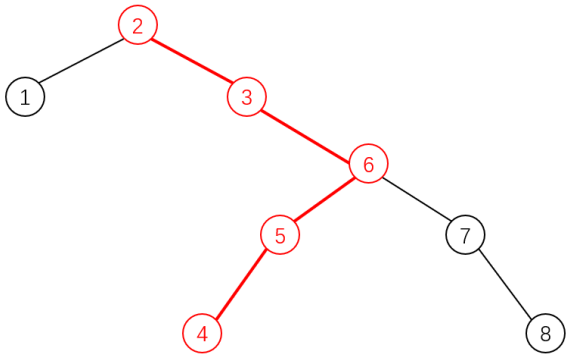
Exercise 6

If $\mathbf{E}[B_{i,j,k}] = 1$, then k is the ancestor of i and j.

Considering the Lemma of $A_{i,j}$:$A_{i,j} = 1$ if and only if among the elements of $[i : j]$, element i comes first in the input array.

So, $B_{i,j,k} = 1$ if and only if i comes first in $[i, j]$ and $[i, k]$.

The length of section is $\max(i,j,k) - \min(i,j,k)$, so $B_{i,j,k} = \dfrac{1}{\max(i,j,k) - \min(i,j,k) + 1}$.



quicksort tree of [2, 3, 6, 1, 7, 5, 8, 4]

Figure 1: The quicksort tree in Exercise 5

$$\mathbf{E}(B_{i,j,k}) = \frac{1}{\max(i,j,k) - \min(i,j,k) + 1}$$

Exercise 7

$$C(\pi, k) = \sum_{i \neq j} B_{i,j,k}$$

Exercise 8

$$\begin{aligned}
\mathbf{E}(C(\pi,1)) &= \mathbf{E}\left(\sum_{i\neq j} B_{i,j,1}\right) \\
&= \sum_{i\neq j} \frac{1}{\max(i,j)} \\
&= 2n - H(n) \\
&= 2n - \ln n + o(1)
\end{aligned}$$

Exercise 9

$$\begin{aligned}
\mathbf{E}_\pi[C(\pi,k)] &= \sum_{i\neq j} \frac{1}{\max(i,j,k) - \min(i,j,k) + 1} \\
&= 2n - H(k) - H(n-k+1) + 2\sum_{i=1}^{n-k} (H(k+i) - H(i+1)) \\
&= 2\left(n + \ln \frac{n!}{k!(n-k)!} - \ln k - \ln(n-k+1)\right) + o(n)
\end{aligned}$$

By the Stirling's formula, we have

$$\begin{aligned}\mathbf{E}_\pi[C(\pi,k)] &= 2\left(n + n\ln n - k\ln k - (n-k)\ln(n-k)\right) + o(n)\\ &= 2n(1 - \kappa\ln\kappa - (1-\kappa)\ln(1-\kappa)) + o(n)\end{aligned}$$