Data Structures and Algorithms (INFO-F413)
Assignment 2: Randomized Selection

Charlotte Nachtegael

October 15, 2016

1 Source code of the program in Python 3

```python
from math import log
from random import sample, randint


def create_list(n):
    """
    Create a list of n unsorted elements belonging to the interval [0,n*2[
    :param n: integer, number of elements for the list
    :return: list of n unsorted elements
    """
    return sample(range(n * 2), n)


def quick_select(list_to_sort, k, count):
    """
    Select the kth smallest element of a unsorted list by using the principle of the Quick Sort algorithm.
    :param list_to_sort: unsorted list of integers
    :param k: rank of the element we desired to find in the unsorted list (1 to the size of the list)
    :param count: number of comparisons done until now, integer
    :return: the kth smallest element and the number of comparisons done to find it
    """
    n = len(list_to_sort)

    # select a random element of the list and class all the others according to it
    pivot = list_to_sort[randint(0, n - 1)]
    list_to_sort.remove(pivot)
    list_under = []
    list_upper = []

    for element in list_to_sort:
        count += 1
        if element < pivot:
```

```
            list_under.append(element)
        elif element > pivot:
            list_upper.append(element)

    # determine if the kth smallest element is the pivot, in the list of elements smaller than
    # the pivot or bigger than the pivot
    if len(list_under) == k - 1:
        res = count, pivot

    elif len(list_under) > k - 1:
        res = quick_select(list_under, k, count)

    else:
        res = quick_select(list_upper, k - len(list_under) - 1, count)

    return res


def summary(count, n, k):
    """
    Compare between the observed average number of comparisons and the upper bound of expected number of comparisons, print the results and confirm if the upper bound is respected
    :param count: total number of comparisons for all the runs
    :param n: size of the list
    :param k: rank of the element we searched for
    """
    average = count / 1000
    expected_count = 2 * n + 2 * n * log(n / (n - k)) + 2 * k * log((n - k) / k)

    if average < expected_count:
        print("The above bound is respected with a average of %s against the expected number of comparisons of %s." % (average, expected_count))
    else:
        print("Error")


def main(n, k):
    """
    Run the Quick Select Algorithm 1000 times with a specified size for the list and rank to find, and confirm if it respects the upper bound of the number of comparisons
    :param n: integer, size of the list
    :param k: integer, rank of the smallest element of the list to look for
    """
    count = 0
    for i in range(1000):
        list_to_sort = create_list(n)
        # print(list_to_sort)
        newcount, element = quick_select(list_to_sort, k, 0)
```

```
        # print(element)
        count += newcount
    summary(count, n, k)


if __name__ == "__main__":
    n = int(input("Size of the list : "))
    k = int(input("Which element do you want to find ? "))
    main(n, k)
```

2 Observation of the results obtained

The program was run with different sizes (n) for the unsorted list and different ranks (k) for the smallest element we are looking for. For each combination of n/k, the program ran 1000 times, each time with a new randomly generated unsorted list to take off the input bias.
We compared the average number of comparisons done during these 1000 times with the expected number of comparisons.

n	k	Expected number of comparisons	Average number of comparisons
5	3	16.730116670092567	6.748
5	4	15.004024235381879	6.258
10	2	30.008048470763757	16.373
10	3	32.21728604109787	18.1
10	4	33.46023334018513	18.883
10	5	33.86294361119891	19.564
10	6	33.460233340185134	19.363
10	7	32.217286041097864	18.634
10	8	30.008048470763757	17.667
10	9	26.501659467828972	16.236
20	2	53.00331893565794	35.933
20	5	62.493405784752326	42.613
20	10	67.72588722239782	47.986
20	15	62.49340578475234	44.968
100	25	312.46702892376163	281.88
100	50	338.6294361119891	306.028
100	75	312.46702892376163	285.925
1000	250	3124.6702892376165	3115.279
1000	500	3386.2943611198907	3383.387
1000	750	3124.670289237617	3034.499

We observed that the upper bound is always respected : the average number of comparisons is beneath the expected number of comparisons.

When we increase the size of the list, we also increase both the expected number of comparisons, but also the average number. This is logical because the bigger the list, the longer you take to find a specific element inside it.

Another observation is that the expected number and the observed average number of comparisons is maximum when we look for the median of the list. We also saw that the expected number and the average number of comparisons are more or so symmetric from each side of the median. With a $k = \frac{n}{4}$ and a $k = \frac{3n}{4}$, we can observe the same expected number of comparisons and close average number of comparisons. This is due to the two opposites functions composing the expected number of comparisons.