VE281 Homework II

Performance Analysis for Selection Algorithms

Name: Tianyi Ge **Stu Number:** 516370910168

1 Introduction

In this report, we discuss the time complexity of three types of different selection algorithms including Randomized selection, Deterministic selection and Quick sorts, by testing their average runtime with input in different sizes. For each test case, every position will be tested as the order we want. The graph of time-size relationship, intuitively demonstrates the time complexity, thus confirming our theoretical expectation.

Since here we have to test each position as a test case, the time complexity for testing is raised to $O(n^2)$ and $O(n^2 \log n)$ respectively. Thus, the maximum test case size is limited to 6×10^4, though the pattern is still clearly demonstrated.

2 Performance analysis on Selection Algorithms

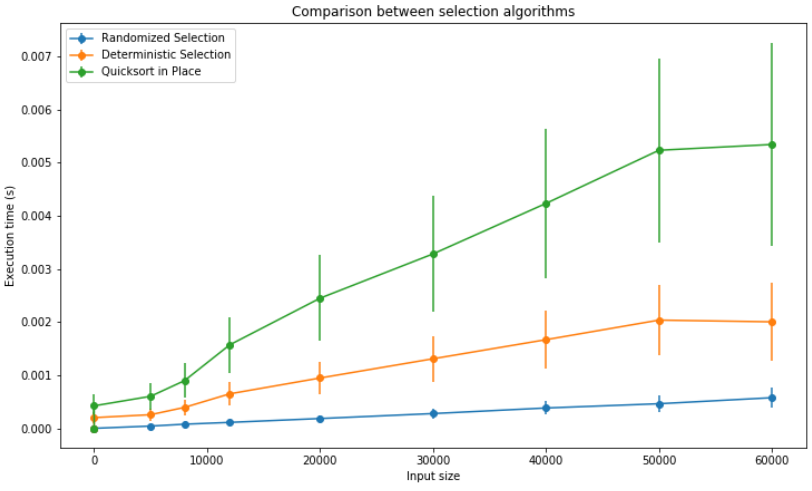

Figure 1: Performance of selection algorithms

To graphically demonstrate the outcome, here the standard deviation of tests are remained as error bar. The figure, as we expected, articulate a lower efficiency of the Quicksort solution.

Theoretically, in the worst case, Randomized selection will fail every time and degrade itself to the worst case of Quicksort, with time complexity $O(n^2)$. For Deterministic selection, the worst case is effectively restricted to $O(n)$ due to the `ChoosePivot` procedure.

For the two $O(n)$ selection algorithms, Randomized selection obtains a conspicuous advantage both on the runtime and standard deviation. Its little vibration indicates the stableness and reliability. In comparison, the uncertainty of Deterministic and Quicksort runtime is enormous.

3 Conclusion

In this report, we have demonstrated the characteristics of three different selection algorithms as well as their performance. Randomized selection is essentially the simplified version of Quicksort. Despite its unreliable worst case, it still obtains a better performance on average among these three algorithms. However, Deterministic selection makes a trade-off, which sacrifices its average performance for limiting its upper bound. In practice, it's sometimes worth a try and guarantees the steadiness of an entire procedure.

4 Appendix

A Source Codes

Program code 1: Three selection algorithms

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
inline void swap(int &a, int &b) {
    int t = a; a = b; b = t;
}

int partition(int d[], int n, int p) {
    swap(d[0], d[p]);
    int key = d[0];
    int i = 0, j = n - 1;
    while (i < j) {
        while(d[j] >= key && i < j) --j;
        while(d[i] <= key && i < j) ++i;
        if (i < j) swap(d[i], d[j]);
    }
    d[0] = d[i];
    d[i] = key;
    return i;
}

void insertionsort(int d[], int n) {
    for (int i = 1; i < n; ++i) {
        int key = d[i], j;
        for (j = 0; j < i && d[j] <= key; ++j);
        for (int k = i; k > j; --k) d[k] = d[k-1];
        d[j] = key;
    }
}

void medians(int d[], int n) {
    int cnt = 0;
```

```
34      for (int i = 0; i + 5 < n; i += 5) {
35          insertionsort(d+i, 5);
36          swap(d[cnt++], d[i+2]);
37      }
38  }
39
40  int Rselect(int d[], int n, int k) {
41      if (n == 1) return d[0];
42      int p = rand() % n;
43      int j = partition(d, n, p);
44      if (j == k) return d[j];
45      if (j > k) return Rselect(d, j, k);
46      else return Rselect(d+j+1, n-j-1, k-j-1);
47  }
48
49  int Dselect(int d[], int n, int k) {
50      if (n == 1) return d[0];
51      if (n <= 5) {
52          insertionsort(d, n);
53          return d[k];
54      }
55      medians(d, n);
56      Dselect(d, n/5, n/10);
57      int p = n/10;
58      int j = partition(d, n, p);
59      if (j == k) return d[j];
60      if (j > k) return Dselect(d, j, k);
61      else return Dselect(d+j+1, n-j-1, k-j-1);
62  }
63
64  void quicksort_inplace(int *d, int l, int r) {
65      if (l >= r) return;
66      int p = rand()%(r-l+1)+l;
67      swap(d[l], d[p]);
68      int key = d[l];
69      int i = l, j = r;
70      while (i < j) {
71          while(d[j] >= key && i < j) --j; // make sure finally i == j and
            ↪  d[j]=d[i] < key so that you could put it on the left
```

```
72          while(d[i] <= key && i < j) ++i;
73          if (i < j) swap(d[i], d[j]);
74      }
75      d[l] = d[i];
76      d[i] = key;
77      quicksort_inplace(d, l, i - 1);
78      quicksort_inplace(d, i + 1, r);
79  }
80
81
82  int main() {
83      ios::sync_with_stdio(false);
84      srand(time(NULL));
85      int n, k, flag;
86      cin >> flag >> k >> n;
87      int *d = new int[n];
88      for (int i = 0; i < n; ++i) cin >> d[i];
89      int start = clock();
90      if (flag == 0) Rselect(d, n, k);
91      else if (flag == 1) Dselect(d, n, k);
92      else quicksort_inplace(d, 0, n-1);
93      cout << (clock() - start)*1.0/CLOCKS_PER_SEC << "\n";
94      delete[] d;
95      return 0;
96  }
```

Program code 2: Test case generator

```
1   #!/usr/bin/python
2
3   import sys
4
5   import random
6   import os
7   import time
8
9   MAX = 10
10  INT_MAX = 2**31;
```

```python
size = [1, 5, 5000, 8000, 12000, 20000, 30000, 40000, 50000, 60000]
if __name__ == "__main__":
    for cases in range(MAX):
        filename = '../inputs/{}.in'.format(cases)
        dirname = os.path.dirname(filename)
        if not os.path.exists(dirname):
            os.makedirs(dirname)
        with open(filename, 'w') as w:
            n = size[cases]
            w.write(str(n) + '\n')
            for i in range(n):
                w.write(str(random.randint(-INT_MAX, INT_MAX-1)) + '\n')
        print("Cases{}".format(cases))
```

Program code 3: Cases runner

```zsh
#!/bin/zsh
size=(1 5 5000 8000 12000 20000 30000 40000 50000 60000)
for ((i=0; i<3; i++)); do
    for ((j=0; j<10; j++)); do
        s=${size[$j+1]}
        for ((k=0; k<s; k++)); do
            echo $i $k | ./test< ../inputs/$j.in >> ../outs/$i$j.out
        done
    done
done
```

Program code 4: Plotting program

```python
import matplotlib.pyplot as plt
from scipy.stats import t
import numpy as np
MAX = 10
size = [1, 5, 5000, 8000, 12000, 20000, 30000, 40000, 50000, 60000]
plt.figure(figsize=(12,7))
for flag in range(3):
    y=np.array([])
```

```
9       conf_interval=np.array([])
10      for cases in range(MAX):
11          with open('../results/{}{}.out'.format(flag, cases), 'r') as f:
12              data = f.read();
13              time = data.split('\\n')
14              time = time[:-1]
15              time = np.array(time)
16              time = [float(i) for i in time]
17              time = np.array(time)
18
19              y = np.append(y, np.mean(time))
20              n = y.size
21              mean = np.mean(y)
22              std = np.std(y)
23              conf_interval = np.append(conf_interval, std)
24      plt.errorbar(size, y, yerr = conf_interval, fmt = '-o')
25  plt.legend(['Randomized Selection', 'Deterministic Selection', 'Quicksort in
    ↪ Place'], loc = 'upper left')
26  plt.xlabel('Input size')
27  plt.ylabel('Execution time (s)')
28  plt.title('Comparison between selection algorithms')
29  plt.savefig('res.png')
30  plt.show()
```