

# FCE-hw2

Weikun Su

NUID: 001498198

October 2019

## Question 1 (Randomized Quicksort)

The results are shown below.

```
0 The original array is:
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
2 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
3 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
4 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
5 89 90 91 92 93 94 95 96 97 98 99 100
6 The running time of trial 1 is: 1e-005 seconds
The running time of trial 2 is: 9.3e-006 seconds
The running time of trial 3 is: 9.1e-006 seconds
The running time of trial 4 is: 9e-006 seconds
The running time of trial 5 is: 8e-006 seconds
```

The code is shown below.

```
0 //Quick Sort
1 #include <iostream>
2 #include <windows.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 int partition(int v[], int p, int q){
7     int pivot = v[p];
8     int i = p;
9     int j;
10    for (j = p+1; j <= q; j++){
11        if (v[j] < pivot){
12            i++;
13            std::swap(v[j], v[i]);
14        }
15    }
16    std::swap(v[i], v[p]);
17    return i;
18 }
19 int partition_rand(int v[], int p, int q){
20     int pivot_ind;
21     pivot_ind = rand() % (q-p+1) + p;
22     std::swap(v[p], v[pivot_ind]);
23     return partition(v, p, q);
24 }
25 void quickSort_rand(int v[], int p, int q){
26     int r;
27     if (p < q){
```

```

28         r = partition_rand(v, p, q);
        quickSort_rand(v, p, r-1);
30         quickSort_rand(v, r+1, q);
    }
32 }

34 int main(){
    LARGE_INTEGER nFreq;
36    LARGE_INTEGER nBeginTime;
    LARGE_INTEGER nEndTime;
38    QueryPerformanceFrequency(&nFreq); // get the frequency of the counter
    double t;
40    int j;
    int N = 100;
42    int v[N];
    int i;
44    srand(time(NULL));
    for (i=0; i<N; i++){
46        v[i] = i+1;
    }
48    std::cout << "The original array is:" << std::endl; // print original array
    for (i=0; i<N; i++){
50        std::cout << v[i] << '\t';
    }
52    std::cout << std::endl;
    for (j=0; j<5; j++){
54        QueryPerformanceCounter(&nBeginTime); //begin time of quick sort
        quickSort_rand(v, 0, N-1);
56        QueryPerformanceCounter(&nEndTime); //end time of quick sort
        t = (double)(nEndTime.QuadPart-nBeginTime.QuadPart)/(double)nFreq.QuadPart; //
        running time of quick sort
58        std::cout << "The running time of trial " << j+1 << " is: " << t << " seconds"
        << std::endl;
    }
60
    return 0;
62 }

```

## Question 2 (Heapsort)

Result:

```

0 Original array:
92 20 14 97 47 12 24 66 64 41 42 43 76 91 69 77 13 34 81 36 33 87
   70 61 68 4 57 5 84 31 26 55 15 83 52 18 95 9 49 67 50 19 62 78
   11 79 32 75 35 53 22 71 74 51 29 23 82 2 54 96 90 30 10 44 6
   100 25 40 98 56 94 65 7 99 16 58 80 63 1 38 17 60 27 21 88 86 28
   59 85 48 37 46 39 8 73 72 89 93 45 3
2 Sorted array:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
  27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
  48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
  68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
  89 90 91 92 93 94 95 96 97 98 99 100
4 The running time is: 1.41e-005 seconds

```

Code:

```

0 //Heap Sort

```

```

1 #include <iostream>
2 #include <windows.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 void randperm(int a[], int N){
7     int i, j;
8     for (i=0; i<N; i++){
9         j = rand() % (N-i) + i;
10        std::swap(a[i], a[j]);
11    }
12 }
13
14 void max_heapify(int a[], int N, int i){
15     int left, right, largest;
16     left = 2*i + 1;
17     right = 2*i + 2;
18     if (left < N && a[left] > a[i]){
19         largest = left;
20     }
21     else
22     {
23         largest = i;
24     }
25     if (right < N && a[right] > a[largest]){
26         largest = right;
27     }
28     if (largest != i){
29         std::swap(a[i], a[largest]);
30         max_heapify(a, N, largest);
31     }
32 }
33
34 void build_max_heap(int a[], int N){
35     int i;
36     for (i = N/2 - 1; i >= 0; i--){
37         max_heapify(a, N, i);
38     }
39 }
40
41 void heap_sort(int a[], int N){
42     build_max_heap(a, N);
43     int i;
44     for (i = N - 1; i >= 0; i--){
45         std::swap(a[i], a[0]);
46         max_heapify(a, i, 0);
47     }
48 }
49
50 int main(){
51     int i;
52     int N = 100;
53     int v[N];
54     LARGE_INTEGER nFreq;
55     LARGE_INTEGER nBeginTime;
56     LARGE_INTEGER nEndTime;
57     QueryPerformanceFrequency(&nFreq); // get the frequency of the counter
58     double t;
59     srand(time(NULL));
60     for (i=0; i<N; i++){
61         v[i] = i+1;
62     }
63     randperm(v, N);
64     std::cout << "Original array:" << std::endl;
65     for (i=0; i<N; i++){
66         std::cout << v[i] << '\t' ;
67     }
68     std::cout << std::endl;
69     QueryPerformanceCounter(&nBeginTime);

```

```

66     heap_sort(v, N);
    QueryPerformanceCounter(&nEndTime);
68     t = (double)(nEndTime.QuadPart-nBeginTime.QuadPart)/(double)nFreq.QuadPart;
    std::cout << "Sorted array:" << std::endl;
70     for (i=0; i<N; i++){
        std::cout << v[i] << '\t' ;
72     }
    std::cout << std::endl;
74     std::cout << "The running time is: " << t << " seconds" << std::endl;
    return 0;
76 }

```

### Question 3 (Counting Sort)

Result:

```

0 Original array is:
20 18 5 7 16 10 9 3 12 14 0
2 Sorted array is:
0 3 5 7 9 10 12 14 16 18 20
4 The running time is: 5e-007 seconds

```

Code:

```

0 //Counting sort
#include <iostream>
2 #include <windows.h>
#include <algorithm>
4
void counting_sort(int a[], int n){
6     int k;
    k = *std::max_element(a, a + n);
8     int i;
    int c[k+1];
10    int b[n];
    for (i = 0; i <= k; i++){
12        c[i] = 0;
    }
14    for (i = 0; i < n; i++){
        c[a[i]] = c[a[i]] + 1;
16    }
    for (i = 1; i <= k; i++){
18        c[i] = c[i] + c[i-1];
    }
20    for (i = n-1; i >= 0; i--){
        b[c[a[i]]-1] = a[i];
22        c[a[i]] = c[a[i]]-1;
    }
24    for (i = 0; i < n; i++){
        a[i] = b[i];
26    }
}
28 int main(){
    LARGE_INTEGER nFreq;
30    LARGE_INTEGER nBeginTime;
    LARGE_INTEGER nEndTime;
32    QueryPerformanceFrequency(&nFreq); // get the frequency of the counter
    double t;
34    int a[] = {20,18,5,7,16,10,9,3,12,14,0};

```

```

36     int n;
    n = sizeof(a)/sizeof(a[0]);
    int i;
38     std::cout << "Original array is:" << std::endl;
    for (i = 0; i < sizeof(a)/sizeof(a[0]); i++){
40         std::cout << a[i] << '\t';
    }
42     std::cout << std::endl;
    QueryPerformanceCounter(&nBeginTime);
    counting_sort(a, n);
44     QueryPerformanceCounter(&nEndTime);
    t = (double)(nEndTime.QuadPart-nBeginTime.QuadPart)/(double)nFreq.QuadPart;
46     std::cout << "Sorted array is:" << std::endl;
    for (i = 0; i < sizeof(a)/sizeof(a[0]); i++){
48         std::cout << a[i] << '\t';
    }
50     std::cout << std::endl;
52     std::cout << "The running time is: " << t << " seconds" << std::endl;
}

```

## Question 4 (Radix Sort)

Result:

```

0 Original array is:
  329 457 657 839 436 720 353
2 Sorted array is:
  329 353 436 457 657 720 839
4 The running time is: 1.3e-006 seconds

```

Code:

```

0 //Radix Sort
#include <iostream>
2 #include <windows.h>
#include <algorithm>
4
void counting_sort_d(int a[], int n, int digit){
6     int b[n];
    int c[10];
8     int i;
    for (i = 0; i < 10; i++){
10         c[i] = 0;
    }
12     for (i = 0; i < n; i++){
        c[a[i]/digit % 10] = c[a[i]/digit % 10] + 1;
14     }
    for (i = 1; i < 10; i++){
16         c[i] = c[i] + c[i-1];
    }
18     for (i = n-1; i >= 0; i--){
        b[c[a[i]/digit % 10]-1] = a[i];
20         c[a[i]/digit % 10] = c[a[i]/digit % 10] - 1;
    }
22     for (i = 0; i < n; i++){
        a[i] = b[i];
24     }
}
26 void radix_sort(int a[], int n){

```

```

28     int digit;
    for (digit = 1; digit <= 100; digit = digit*10){
        counting_sort_d(a, n, digit);
30     }
}
32 int main(){
    LARGE_INTEGER nFreq;
34     LARGE_INTEGER nBeginTime;
    LARGE_INTEGER nEndTime;
36     QueryPerformanceFrequency(&nFreq); // get the frequency of the counter
    double t;
38     int a[] = {329,457,657,839,436,720,353};
    int n;
40     int i;
    n = sizeof(a)/sizeof(a[0]);
42     std::cout << "Original array is:" << std::endl;
    for (i = 0; i < n; i++){
44         std::cout << a[i] << '\t';
    }
46     std::cout << std::endl;
    QueryPerformanceCounter(&nBeginTime);
48     radix_sort(a, n);
    QueryPerformanceCounter(&nEndTime);
50     t = (double)(nEndTime.QuadPart-nBeginTime.QuadPart)/(double)nFreq.QuadPart;
    std::cout << "Sorted array is:" << std::endl;
52     for (i = 0; i < n; i++){
        std::cout << a[i] << '\t';
54     }
    std::cout << std::endl;
56     std::cout << "The running time is: " << t << " seconds" << std::endl;
}

```