

## **Homework 2 (5pt. + extra 1pt.)**

Submission instruction:

Submit one single pdf file for this homework including both coding problems and analysis problems.

For coding problems, copy and paste your codes. Report your results.

For analysis problems, either type or hand-write and scan.

**Question 1 (2 pt.) Randomized Quicksort:** Write codes for randomized quicksort. You may need `rand()` to generate random numbers. Run the randomized quicksort 5 times for input array  $A = \{1, 2, 3, \dots, 99, 100\}$  and report the 5 running times.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
bash-4.2$ g++ -std=c++11 randomized quicksort.cc -o qsort && ./qsort
Duration to sort trial 1: 133209 ns
Duration to sort trial 2: 128537 ns
Duration to sort trial 3: 128349 ns
Duration to sort trial 4: 128271 ns
Duration to sort trial 5: 128254 ns
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
bash-4.2$

```

**Question 2 (2pt.) Heapsort:** Write codes for heapsort. The input array is a random permutation of  $A = \{1, 2, 3, \dots, 99, 100\}$ . You should write codes to generate and print the random permutation first.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
bash-4.2$ g++ -std=c++11 heapsort.cc -o heap && ./heap
Original Array: 11 27 82 1 98 16 75 77 40 79 91 43 100 67 25 13 33 84 80 50 15 90 85 71 37 42 20 86 10 56 74 44 32 8
1 97 26 60 72 94 78 61 93 12 19 49 55 23 45 3 53 35 62 63 39 69 46 29 38 14 65 51 57 7 30 96 6 47 17 52 41 34 89 92
54 99 28 18 2 31 4 87 95 70 73 36 22 83 21 5 66 88 68 58 8 24 59 76 48 64 9
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
bash-4.2$

```

**Question 3 (1pt.) Counting Sort:** Write codes for counting sort. The input array is  $A = \{20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0\}$ .

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
bash-4.2$ g++ -std=c++11 countingsort.cc -o count && ./count
Original Array: 20 18 5 7 16 10 9 3 12 14 0
Sorted Array: 0 3 5 7 9 10 12 14 16 18 20
bash-4.2$

```

**Question 4 (extra 1pt.) Radix Sort:** Write codes for radix sort: use counting sort for decimal digits from the low order to high order. The input array is  $A = \{329, 457, 657, 839, 436, 720, 353\}$ .

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
bash-4.2$ g++ -std=c++11 radixsort.cc -o radix && ./radix
Original Array: 329 457 657 839 436 720 353
Sorted Array: 329 353 436 457 657 720 839
bash-4.2$

```

```

1  /// Randomized Quicksort ///
2  /// By Anna DeVries      ///
3
4  #include <iostream>
5  #include <stdlib.h>
6  #include <chrono>
7
8  // Rearranges the subarray A[p...r] in place
9  int partition(int A[], int p, int r){
10     int x = A[r];          // pivot point
11     int i = p - 1;
12     int j,hold;
13
14     for(j=p;j<=(r-1);j++){
15         if(A[j] <= x){
16             i = i + 1;
17             std::swap(A[i],A[j]);
18         }
19     }
20
21     std::swap(A[i+1],A[r]);
22     return i+1;
23 }
24
25 // Randomizes pivot point
26 int randomized_partition(int A[], int p, int r){
27     srand(time(NULL));
28     int i = p+rand() % (r-p);
29     std::swap(A[r],A[i]);
30     return partition(A, p, r);
31 }
32
33 // Sorting operation
34 int randomized_quicksort(int A[], int p, int r){
35     if(p < r){
36         int q = randomized_partition(A,p,r);
37         randomized_quicksort(A,p,q-1);
38         randomized_quicksort(A,q+1,r);
39     }
40 }
41
42 int main(){
43     // Initialize array A[1,2,...,99,100]
44     int A[100];
45     for(int i=0;i<100;i++){
46         A[i] = i+1;
47     }
48     int p = 0;
49     int r = sizeof(A)/sizeof(A[0])-1;
50
51     // Perform sorting operations 5 times
52     for(int i=0;i<5;i++){
53         // Initialize clock
54         auto start = std::chrono::high_resolution_clock::now();
55
56         // Perform sorting operations
57         randomized_quicksort(A, p, r);
58
59         // End and print clock
60         auto finish = std::chrono::high_resolution_clock::now();
61         std::chrono::duration<double> elapsed = finish - start;
62         std::cout << "Duration to sort trial "<<i+1<<": " <<
        std::chrono::duration_cast<std::chrono::nanoseconds>(finish - start).count() <<
        " ns\n";
63     }
64
65     // Print new array
66     std::cout<<"Sorted Array: ";
67     for(int j=0;j<sizeof(A)/sizeof(A[0]);j++){

```

```
68         std::cout<<A[j]<<" ";
69     }
70     std::cout<<std::endl;
71 }
```

```

1  //// Heap Sort ////
2  ///By Anna DeVries///
3
4  #include <iostream>
5  #include <stdlib.h>
6  int heapsize,n;
7
8  // Returns left node value
9  int left(int i){
10     return 2*i + 1;
11 }
12
13 // Returns right node value
14 int right(int i){
15     return (2*i)+ 2;
16 }
17
18 // Corrects single instance, ensuring parent key > children keys
19 int max_heapify(int A[], int i){
20     int l = left(i);
21     int r = right(i);
22     int largest;
23
24     if(l < heapsize && A[l] > A[i]){
25         largest = l;
26     }
27     else{
28         largest = i;
29     }
30     if(r < heapsize && A[r] > A[largest]){
31         largest = r;
32     }
33
34     if(largest != i){
35         std::swap(A[i],A[largest]);
36         max_heapify(A,largest);
37     }
38 }
39
40 // Converts array into max_heap
41 int build_max_heap(int A[]){
42     heapsize = n;
43
44     for(int i=(n/2) - 1;i>=0;i--){
45         max_heapify(A,i);
46     }
47 }
48
49 // Sorts array
50 int heapsort(int A[]){
51     build_max_heap(A);
52
53     for(int i=n-1;i>=0;i--){
54         std::swap(A[0],A[i]);
55         heapsize = heapsize - 1;
56         max_heapify(A,0);
57     }
58 }
59
60 int randomize(int A[]){
61     srand(time(NULL));
62     for(int i=n-1;i>0;i--){
63         int j = rand() % (i + 1);
64         int hold = A[i];
65         A[i] = A[j];
66         A[j] = hold;
67     }
68 }
69

```

```
70  int main(){
71      // Initialize array A[1,2,...,99,100] as a random permutation
72      int A[100];
73      for(int i=0;i<100;i++){
74          A[i] = i+1;
75      }
76      n = sizeof(A)/sizeof(A[0]);
77      randomize(A);
78
79      // Print original array
80      std::cout<<"Original Array: ";
81      for(int j=0;j<n;j++){
82          std::cout<<A[j]<<" ";
83      }
84      std::cout<<std::endl;
85
86      // Sorting operation
87      heapsort(A);
88
89      // Print sorted array
90      std::cout<<"Sorted Array: ";
91      for(int j=0;j<n;j++){
92          std::cout<<A[j]<<" ";
93      }
94      std::cout<<std::endl;
95  }
```

```

1  /// Counting Sort ///
2  /// By Anna DeVries ///
3
4  #include <iostream>
5  #include <stdlib.h>
6
7  // Global Variables
8  int n;
9
10 // Sorting operation
11 int counting_sort(int A[],int B[],int k){
12     // Initialize an empty array
13     int C[k+1];
14     for(int i=0;i<(k+1);i++){
15         C[i] = 0;
16     }
17
18     // Count instances of each distinct element
19     for(int j=0;j<n;j++){
20         C[A[j]] = C[A[j]] + 1;
21     }
22
23     // Match C array with size of final array
24     for(int i=1;i<(k+1);i++){
25         C[i] = C[i] + C[i-1];
26     }
27
28     // Sort elements
29     for(int j=(n-1);j>=0;j--){
30         B[C[A[j]]-1] = A[j];
31         C[A[j]] = C[A[j]] - 1;
32     }
33 }
34
35 // Find largest distinct element
36 int distinct(int A[]){
37     int largest = A[0];
38     for(int i=0;i<n;i++){
39         for(int j=0;j<n;j++){
40             if(A[j]>largest){
41                 largest = A[j];
42             }
43             if(A[i]>largest){
44                 largest = A[i];
45             }
46         }
47     }
48     return largest;
49 }
50
51 int main(){
52     // Initialize array
53     int A[] = {20,18,5,7,16,10,9,3,12,14,0};
54
55     n = sizeof(A)/sizeof(A[0]);
56     int B[n-1], k;
57     k = distinct(A);
58
59     // Sorting operation
60     counting_sort(A,B,k);
61
62     // Print results
63     std::cout<<"Original Array: ";
64     for(int j=0;j<n;j++){
65         std::cout<<A[j]<<" ";
66     }
67     std::cout<<std::endl;
68
69     std::cout<<"Sorted Array: ";

```

```
70     for(int j=0;j<n;j++){
71         std::cout<<B[j]<<" ";
72     }
73     std::cout<<std::endl;
74 }
```



```

1  /// Radix Sort ///
2  ///By Anna DeVries///
3
4  #include <iostream>
5  #include <stdlib.h>
6  #include <cmath>
7
8  // Global Variables
9  int n;
10
11 // Sorting operations -- stable sort
12 int counting_sort(int A[],int k){
13     int B[n-1],i;
14
15     // Initialize an empty array
16     int C[10];
17     for(int i=0;i<(10);i++){
18         C[i] = 0;
19     }
20
21     // Count instances of each distinct element
22     for(int j=0;j<n;j++){
23         C[(A[j]/k)%10] = C[(A[j]/k)%10] + 1;
24     }
25
26     // Match C array with size of final array
27     for(int i=1;i<10;i++){
28         C[i] = C[i] + C[i-1];
29     }
30
31     // Sort elements
32     for(int j=(n-1);j>=0;j--){
33         B[C[(A[j]/k)%10]-1] = A[j];
34         C[(A[j]/k)%10] = C[(A[j]/k)%10] - 1;
35     }
36
37     // Copy B[] into A[]
38     for(i=0;i<n;i++){
39         A[i] = B[i];
40     }
41 }
42
43 // Sorting operation -- radix sort
44 int radix_sort(int A[], int d){
45     for(int i=1;d/i>0;i*=10){
46         counting_sort(A,i);
47     }
48 }
49
50 // Find largest distinct element and return its number of digits
51 int distinct(int A[]){
52     int largest = A[0];
53     for(int i=0;i<n;i++){
54         for(int j=0;j<n;j++){
55             if(A[j]>largest){
56                 largest = A[j];
57             }
58             if(A[i]>largest){
59                 largest = A[i];
60             }
61         }
62     }
63     return largest;
64 }
65
66 int main(){
67     // Initialize array
68     int A[] = {329,457,657,839,436,720,353};
69     n = sizeof(A)/sizeof(A[0]);

```

```
70     int d = distinct(A);
71
72     // Print original array
73     std::cout<<"Original Array: ";
74     for(int j=0;j<n;j++){
75         std::cout<<A[j]<<" ";
76     }
77     std::cout<<std::endl;
78
79     // Sorting operation
80     radix_sort(A,d);
81
82     // Print sorted array
83     std::cout<<"Sorted Array: ";
84     for(int j=0;j<n;j++){
85         std::cout<<A[j]<<" ";
86     }
87     std::cout<<std::endl;
88 }
```