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

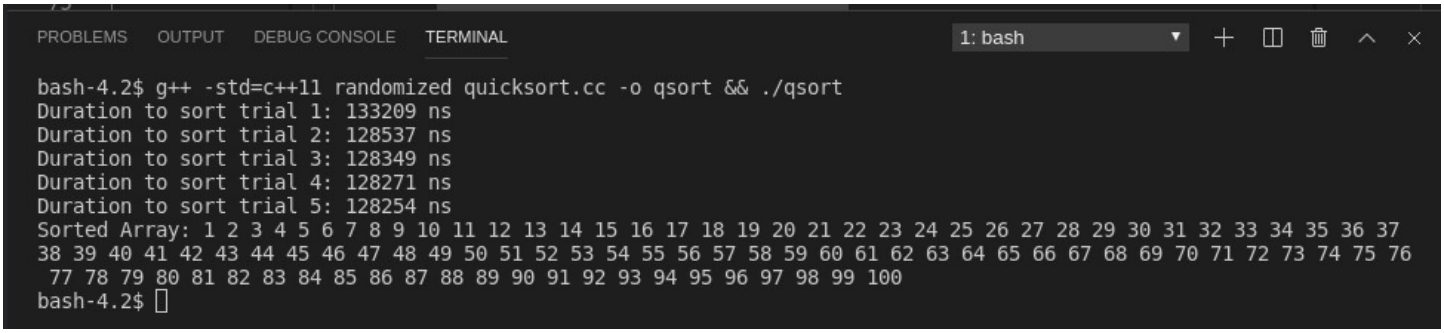
Submission instruction:

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

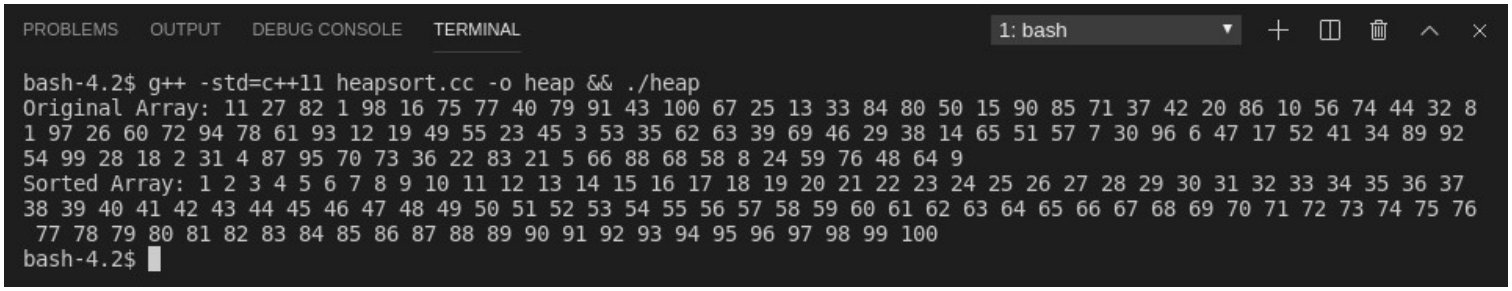
For coding problems, copy and paste your codes. Report your results.

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

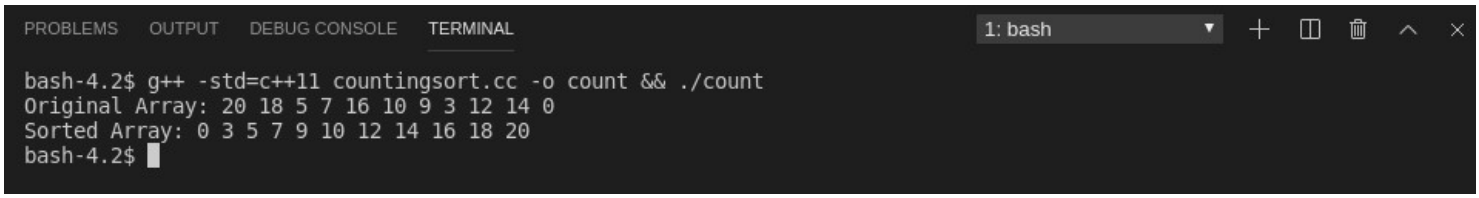
Question 1 (2 pt.) Randomized Quicksort: Write codes for randomized quicksort. You may need rand() to generate random numbers. Run the randomized quicksort 5 times for input array A = {1, 2, 3, …, 99, 100} and report the 5 running times.



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



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}.



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}.

```cpp
//// Randomized Quicksort ////
///  By Anna DeVries      ///

#include <iostream>
#include <stdlib.h>
#include <chrono>

// Rearranges the subarray A[p...r] in place
int partition(int A[], int p, int r){
    int x = A[r];       // pivot point
    int i = p - 1;
    int j,hold;

    for(j=p;j<=(r-1);j++){
        if(A[j] <= x){
            i = i + 1;
            std::swap(A[i],A[j]);
        }
    }

    std::swap(A[i+1],A[r]);
    return i+1;
}

// Randomizes pivot point
int randomized_partition(int A[], int p, int r){
    srand(time(NULL));
    int i = p+rand() % (r-p);
    std::swap(A[r],A[i]);
    return partition(A, p, r);
}

// Sorting operation
int randomized_quicksort(int A[], int p, int r){
    if(p < r){
        int q = randomized_partition(A,p,r);
        randomized_quicksort(A,p,q-1);
        randomized_quicksort(A,q+1,r);
    }
}

int main(){
    // Initialize array A[1,2,...,99,100]
    int A[100];
    for(int i=0;i<100;i++){
        A[i] = i+1;
    }
    int p = 0;
    int r = sizeof(A)/sizeof(A[0])-1;

    // Perform sorting operations 5 times
    for(int i=0;i<5;i++){
        // Initialize clock
        auto start = std::chrono::high_resolution_clock::now();

        // Perform sorting operations
        randomized_quicksort(A, p, r);

        // End and print clock
        auto finish = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> elapsed = finish - start;
        std::cout << "Duration to sort trial "<<i+1<<": " <<
        std::chrono::duration_cast<std::chrono::nanoseconds>(finish - start).count() <<
        " ns\n";
    }

    // Print new array
    std::cout<<"Sorted Array: ";
    for(int j=0;j<sizeof(A)/sizeof(A[0]);j++){
```

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

```
//// Heap Sort ////
///By Anna DeVries///

#include <iostream>
#include <stdlib.h>
int heapsize,n;

// Returns left node value
int left(int i){
    return 2*i + 1;
}

// Returns right node value
int right(int i){
    return (2*i)+ 2;
}

// Corrects single instance, ensuring parent key > children keys
int max_heapify(int A[], int i){
    int l = left(i);
    int r = right(i);
    int largest;

    if(l < heapsize && A[l] > A[i]){
        largest = l;
    }
    else{
        largest = i;
    }
    if(r < heapsize && A[r] > A[largest]){
        largest = r;
    }

    if(largest != i){
        std::swap(A[i],A[largest]);
        max_heapify(A,largest);
    }
}

// Converts array into max_heap
int build_max_heap(int A[]){
    heapsize = n;

    for(int i=(n/2) - 1;i>=0;i--){
        max_heapify(A,i);
    }
}

// Sorts array
int heapsort(int A[]){
    build_max_heap(A);

    for(int i=n-1;i>=0;i--){
        std::swap(A[0],A[i]);
        heapsize = heapsize - 1;
        max_heapify(A,0);
    }
}

int randomize(int A[]){
    srand(time(NULL));
    for(int i=n-1;i>0;i--){
        int j = rand() % (i + 1);
        int hold = A[i];
        A[i] = A[j];
        A[j] = hold;
    }
}

```

```cpp
int main(){
    // Initialize array A[1,2,...,99,100] as a random permutation
    int A[100];
    for(int i=0;i<100;i++){
        A[i] = i+1;
    }
    n = sizeof(A)/sizeof(A[0]);
    randomize(A);

    // Print original array
    std::cout<<"Original Array: ";
    for(int j=0;j<n;j++){
        std::cout<<A[j]<<" ";
    }
    std::cout<<std::endl;

    // Sorting operation
    heapsort(A);

    // Print sorted array
    std::cout<<"Sorted Array: ";
    for(int j=0;j<n;j++){
        std::cout<<A[j]<<" ";
    }
    std::cout<<std::endl;
}
```

```cpp
//// Counting Sort ////
/// By Anna DeVries ///

#include <iostream>
#include <stdlib.h>

// Global Variables
int n;

// Sorting operation
int counting_sort(int A[],int B[],int k){
    // Initialize an empty array
    int C[k+1];
    for(int i=0;i<(k+1);i++){
        C[i] = 0;
    }

    // Count instances of each distinct element
    for(int j=0;j<n;j++){
        C[A[j]] = C[A[j]] + 1;
    }

    // Match C array with size of final array
    for(int i=1;i<(k+1);i++){
        C[i] = C[i] + C[i-1];
    }

    // Sort elements
    for(int j=(n-1);j>=0;j--){
        B[C[A[j]]-1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }
}

// Find largest distinct element
int distinct(int A[]){
    int largest = A[0];
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(A[j]>largest){
                largest = A[j];
            }
            if(A[i]>largest){
                largest = A[i];
            }
        }
    }
    return largest;
}

int main(){
    // Initialize array
    int A[] = {20,18,5,7,16,10,9,3,12,14,0};

    n = sizeof(A)/sizeof(A[0]);
    int B[n-1], k;
    k = distinct(A);

    // Sorting operation
    counting_sort(A,B,k);

    // Print results
    std::cout<<"Original Array: ";
    for(int j=0;j<n;j++){
        std::cout<<A[j]<<" ";
    }
    std::cout<<std::endl;

    std::cout<<"Sorted Array: ";
```

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

```
//// Radix Sort ////
///By Anna DeVries///

#include <iostream>
#include <stdlib.h>
#include <cmath>

// Global Variables
int n;

// Sorting operations -- stable sort
int counting_sort(int A[],int k){
    int B[n-1],i;

    // Initialize an empty array
    int C[10];
    for(int i=0;i<(10);i++){
        C[i] = 0;
    }

    // Count instances of each distinct element
    for(int j=0;j<n;j++){
        C[(A[j]/k)%10] = C[(A[j]/k)%10] + 1;
    }

    // Match C array with size of final array
    for(int i=1;i<10;i++){
        C[i] = C[i] + C[i-1];
    }

    // Sort elements
    for(int j=(n-1);j>=0;j--){
        B[C[(A[j]/k)%10]-1] = A[j];
        C[(A[j]/k)%10] = C[(A[j]/k)%10] - 1;
    }

    // Copy B[] into A[]
    for(i=0;i<n;i++){
        A[i] = B[i];
    }
}

// Sorting operation -- radix sort
int radix_sort(int A[], int d){
    for(int i=1;d/i>0;i*=10){
       counting_sort(A,i);
    }
}

// Find largest distinct element and return its number of digits
int distinct(int A[]){
    int largest = A[0];
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(A[j]>largest){
                largest = A[j];
            }
            if(A[i]>largest){
                largest = A[i];
            }
        }
    }
    return largest;
}

int main(){
    // Initialize array
    int A[] = {329,457,657,839,436,720,353};
    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  }
```