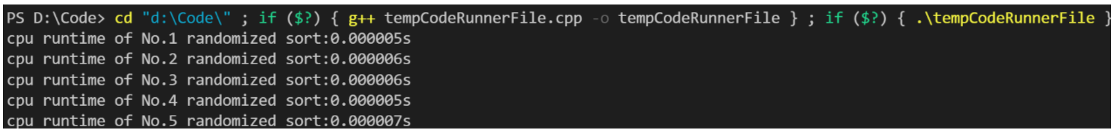
Question 1. Randomized Quicksort: Write codes for randomized quicksort. You may need rand() to generate random numbers. Run the randomized quicksort 5 times for input array A = {1, 2, 3, …, 99, 100} and report the 5 running times.

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>

int randomized_partition(int A[], int start, int end){
    int random_index,m;
    int g, k;
    int tmp;
    random_index = rand() % (end - start + 1) + start;
    m = A[end];
    A[end] = A[random_index];
    A[random_index] = m;
    g = start - 1;
    for(k = start; k <= end - 1; k++){
        if(A[k] <= A[end]){
            g = g + 1;
            tmp = A[g];
            A[g] = A[k];
            A[k] = tmp;
        }
    }
    tmp = A[g + 1];
    A[g + 1] = A[end];
    A[end] = tmp;
    m = g + 1;
    return m;
}

void random_qsort(int A[], int start, int end){
    if(start < end){
        int pivot;
        pivot = randomized_partition(A, start, end);
        random_qsort(A, start, pivot - 1);
        random_qsort(A, pivot + 1, end);
    }
}
```

```
int main(){
    int A[100];
    for(int i = 0; i < 100; i++){
        A[i] = i + 1;
    }
    for(int j = 1; j <= 5; j++){
        LARGE_INTEGER initial_time;
        double cpu_run_time = 0;
        LARGE_INTEGER end_time;
        double f2;
        LARGE_INTEGER f;
        QueryPerformanceFrequency(&f);
        f2 = (double)f.QuadPart;

        QueryPerformanceCounter(&initial_time);
        random_qsort(A, 0, 99);
        QueryPerformanceCounter(&end_time);

        cpu_run_time = ((double)end_time.QuadPart - (double)initial_time.QuadPart) / f2;
        printf("cpu runtime of No.%d randomized sort:%fs\n",j,cpu_run_time);
    }
}
```

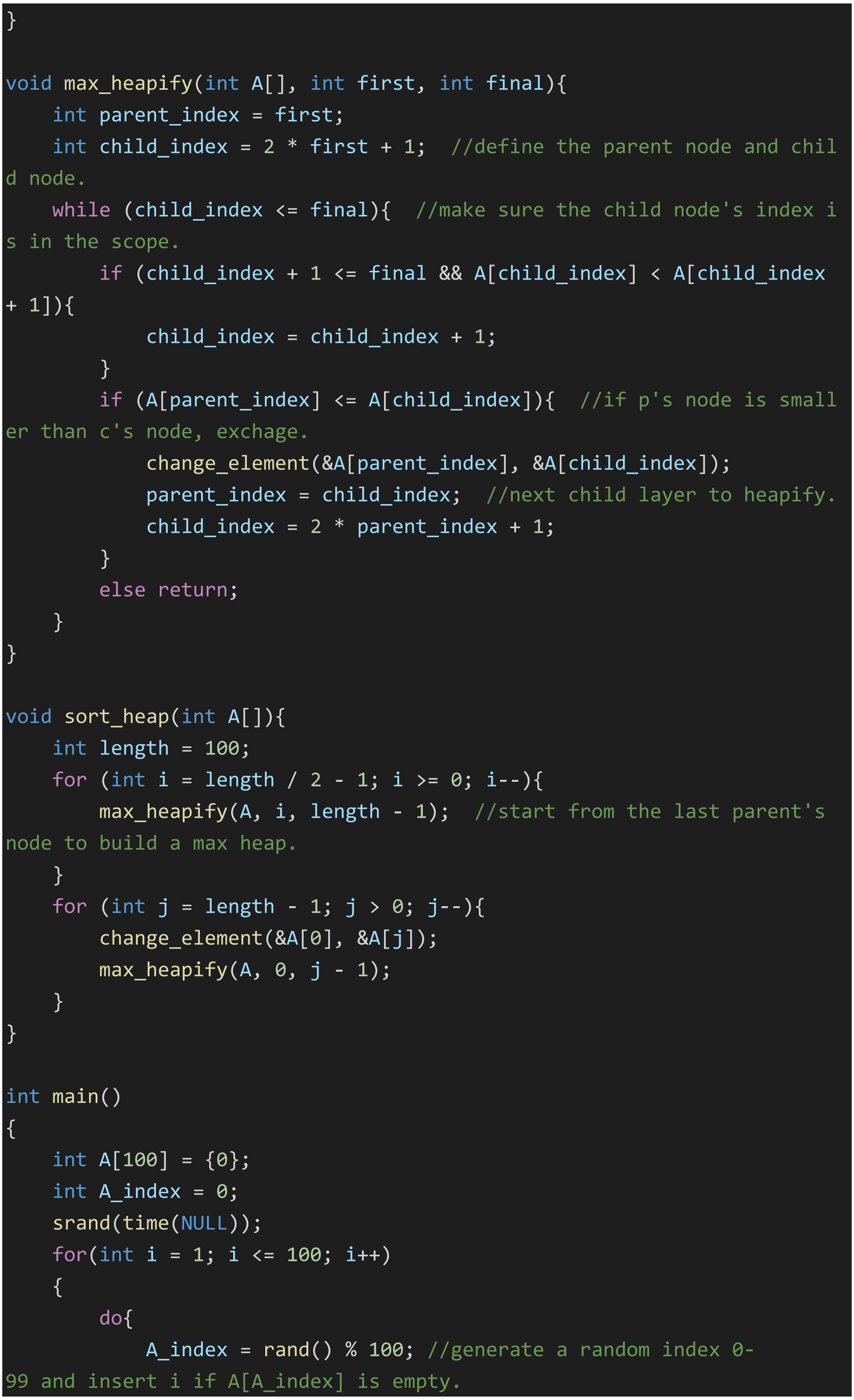
Results:



```
PS D:\Code> cd "d:\Code\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
cpu runtime of No.1 randomized sort:0.000005s
cpu runtime of No.2 randomized sort:0.000006s
cpu runtime of No.3 randomized sort:0.000006s
cpu runtime of No.4 randomized sort:0.000005s
cpu runtime of No.5 randomized sort:0.000007s
```

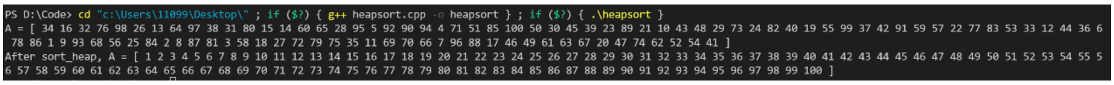
Question 2. Heapsort: Write codes for heapsort. The input array is a random permutation of A = {1, 2, 3, …, 99, 100}. You should write codes to generate and print the random permutation first.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include <math.h>

void change_element(int* x1, int* x2){
    int tmp = *x2;
    *x2 = *x1;
    *x1 = tmp;
```



```c
}

void max_heapify(int A[], int first, int final){
    int parent_index = first;
    int child_index = 2 * first + 1;  //define the parent node and child node.
    while (child_index <= final){  //make sure the child node's index is in the scope.
        if (child_index + 1 <= final && A[child_index] < A[child_index + 1]){
            child_index = child_index + 1;
        }
        if (A[parent_index] <= A[child_index]){  //if p's node is smaller than c's node, exchage.
            change_element(&A[parent_index], &A[child_index]);
            parent_index = child_index;  //next child layer to heapify.
            child_index = 2 * parent_index + 1;
        }
        else return;
    }
}

void sort_heap(int A[]){
    int length = 100;
    for (int i = length / 2 - 1; i >= 0; i--){
        max_heapify(A, i, length - 1);  //start from the last parent's node to build a max heap.
    }
    for (int j = length - 1; j > 0; j--){
        change_element(&A[0], &A[j]);
        max_heapify(A, 0, j - 1);
    }
}

int main()
{
    int A[100] = {0};
    int A_index = 0;
    srand(time(NULL));
    for(int i = 1; i <= 100; i++)
    {
        do{
            A_index = rand() % 100; //generate a random index 0-99 and insert i if A[A_index] is empty.
        }
        while (A[A_index] != 0);
        A[A_index] = i;
    }
    printf("A = [ ");
    for (int i = 0; i < 100; i++)
        printf("%d ",A[i]);
    printf("]");
    printf("\n");

    sort_heap(A);
    printf("After sort_heap, A = [ ");
    for(int k = 0; k < 100; k++){
        printf("%d ",A[k]);
    }
    printf("]");
    printf("\n");
    return 0;
}
```

Results:

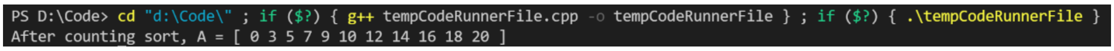


```
PS D:\Code> cd "c:\Users\11099\Desktop\" ; if ($?) { g++ heapsort.cpp -o heapsort } ; if ($?) { .\heapsort }
A = [ 34 16 32 76 98 26 13 64 97 38 31 80 15 14 60 65 28 95 5 92 90 94 4 71 51 85 100 50 30 45 39 23 89 21 10 43 48 29 73 24 82 40 19 55 99 37 42 91 59 57 22 77 83 53 33 12 44 36 6
 78 86 1 9 93 68 56 25 84 2 8 87 81 3 58 18 27 72 79 75 35 11 69 70 66 7 96 88 17 46 49 61 63 67 20 47 74 62 52 54 41 ]
After sort_heap, A = [ 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 5
6 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 ]
```

Question 3. Counting Sort: Write codes for counting sort. The inut array is A = {20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0}.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include <math.h>

void counting_sort(int A[], int length){
    int max_element = 0;
    for(int j = 0; j < length; j++){
        if(A[j] >= max_element){
            max_element = A[j];
        }
        else continue;
    }
    int count[max_element + 1] = {0};
    for(int k = 0; k < length; k++){
        count[A[k]] = count[A[k]] + 1;
    }
    int sig = 0;
    for(int r = 0; r <= max_element; r++){
        if(count[r] == 0) continue;
        else{
            for(int q = 1; q <= count[r]; q++){
                A[sig] = r;
                sig = sig + 1;
            }
        }
    }
}

int main(){
    int A[] = {20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0};
    int length = 11;
    counting_sort(A, length);
    printf("After counting sort, A = [ ");
    for(int i = 0; i < length; i++){
        printf("%d ",A[i]);
    }
    printf("]\n");
    return 0;
}
```

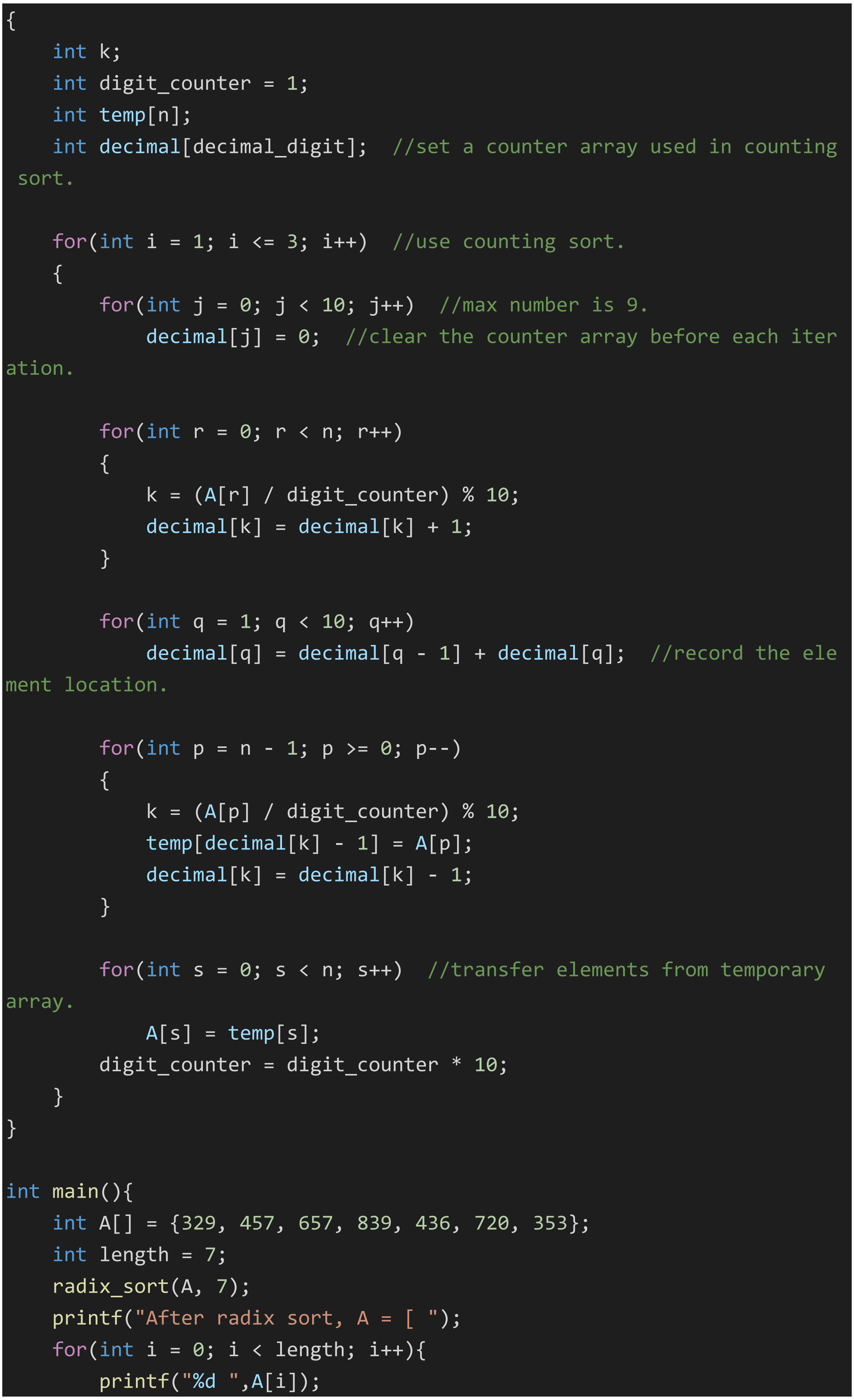
Results:



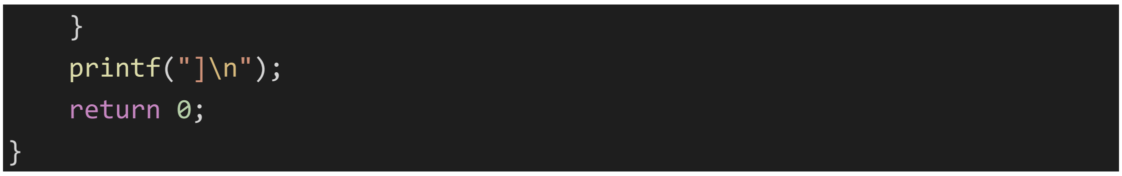
```
PS D:\Code> cd "d:\Code\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
After counting sort, A = [ 0 3 5 7 9 10 12 14 16 18 20 ]
```

Question 4. 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}.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include <math.h>

#define decimal_digit 10

void radix_sort(int A[], int n)
```

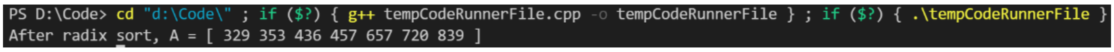


```c
{
    int k;
    int digit_counter = 1;
    int temp[n];
    int decimal[decimal_digit];  //set a counter array used in counting sort.

    for(int i = 1; i <= 3; i++)  //use counting sort.
    {
        for(int j = 0; j < 10; j++)  //max number is 9.
            decimal[j] = 0;  //clear the counter array before each iteration.

        for(int r = 0; r < n; r++)
        {
            k = (A[r] / digit_counter) % 10;
            decimal[k] = decimal[k] + 1;
        }

        for(int q = 1; q < 10; q++)
            decimal[q] = decimal[q - 1] + decimal[q];  //record the element location.

        for(int p = n - 1; p >= 0; p--)
        {
            k = (A[p] / digit_counter) % 10;
            temp[decimal[k] - 1] = A[p];
            decimal[k] = decimal[k] - 1;
        }

        for(int s = 0; s < n; s++)  //transfer elements from temporary array.
            A[s] = temp[s];
        digit_counter = digit_counter * 10;
    }
}

int main(){
    int A[] = {329, 457, 657, 839, 436, 720, 353};
    int length = 7;
    radix_sort(A, 7);
    printf("After radix sort, A = [ ");
    for(int i = 0; i < length; i++){
        printf("%d ",A[i]);
```



```
    }
    printf("]\n");
    return 0;
}
```

Results:



```
PS D:\Code> cd "d:\Code\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
After radix sort, A = [ 329 353 436 457 657 720 839 ]
```