**Assignment # 2**

1. Calculate T(n) and O(n) for the following algorithms (Quick Sort) for the average and worst case:

void **quick\_sort**(int first, int last, std::vector<int>& arr) {

if (last - first > 1) {

// There is data to be sorted.

// Partition the table.

int pivot = partition(first, last,arr);

// Sort the left half.

quick\_sort(first, pivot,arr);

// Sort the right half.

quick\_sort(pivot + 1, last,arr);

}

}

int **partition**(int first, int last, std::vector<int>& arr) {

.

int up = first + 1;

int down = last - 1;

do {

while ((up != last - 1) && arr[first] >= arr[up]) {

++up;

}

while (arr[first] < arr[down]) {

--down;

}

if (up < down) {

// if up is to the left of down,

swap(arr[up],arr[down]);

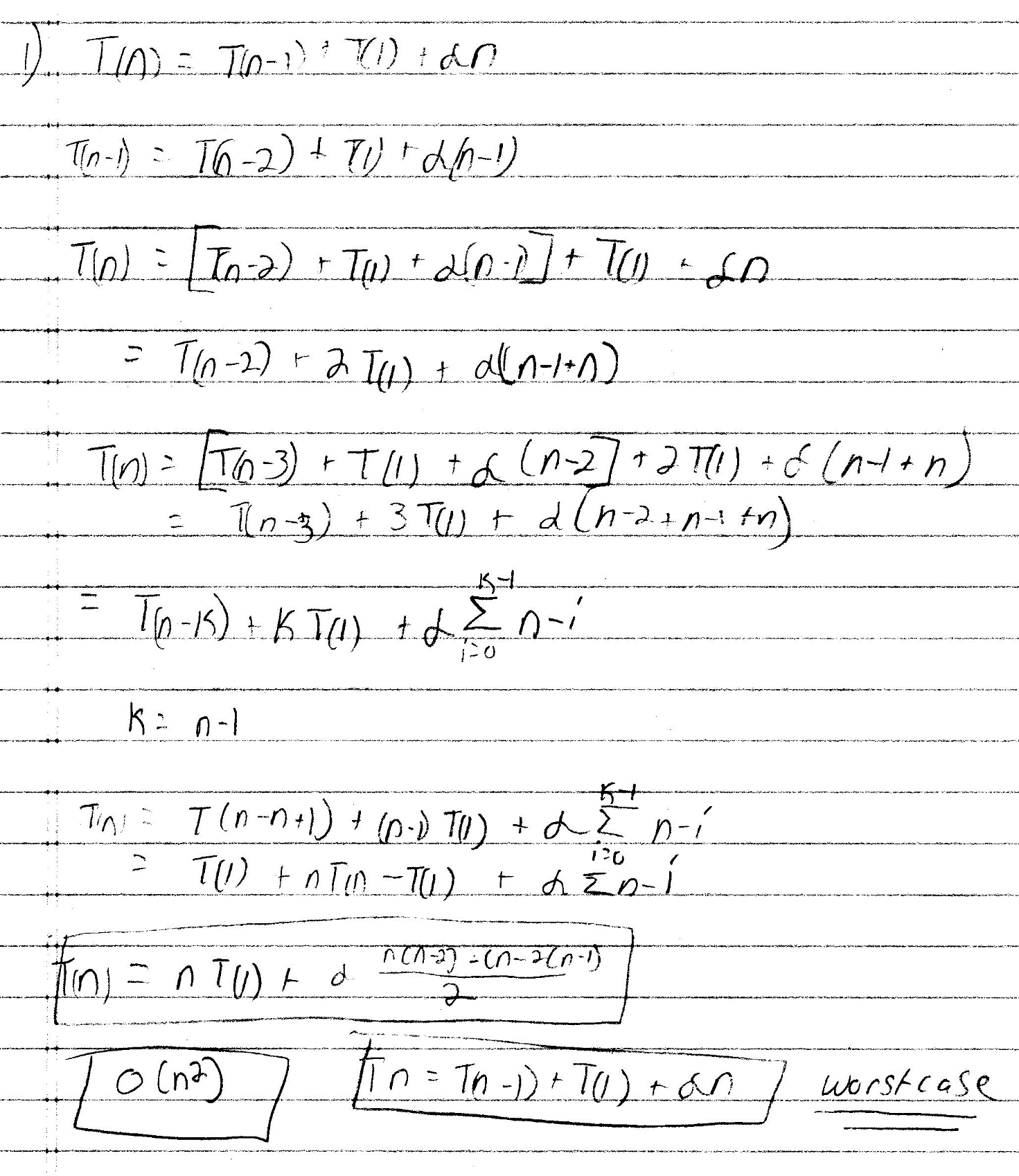
}

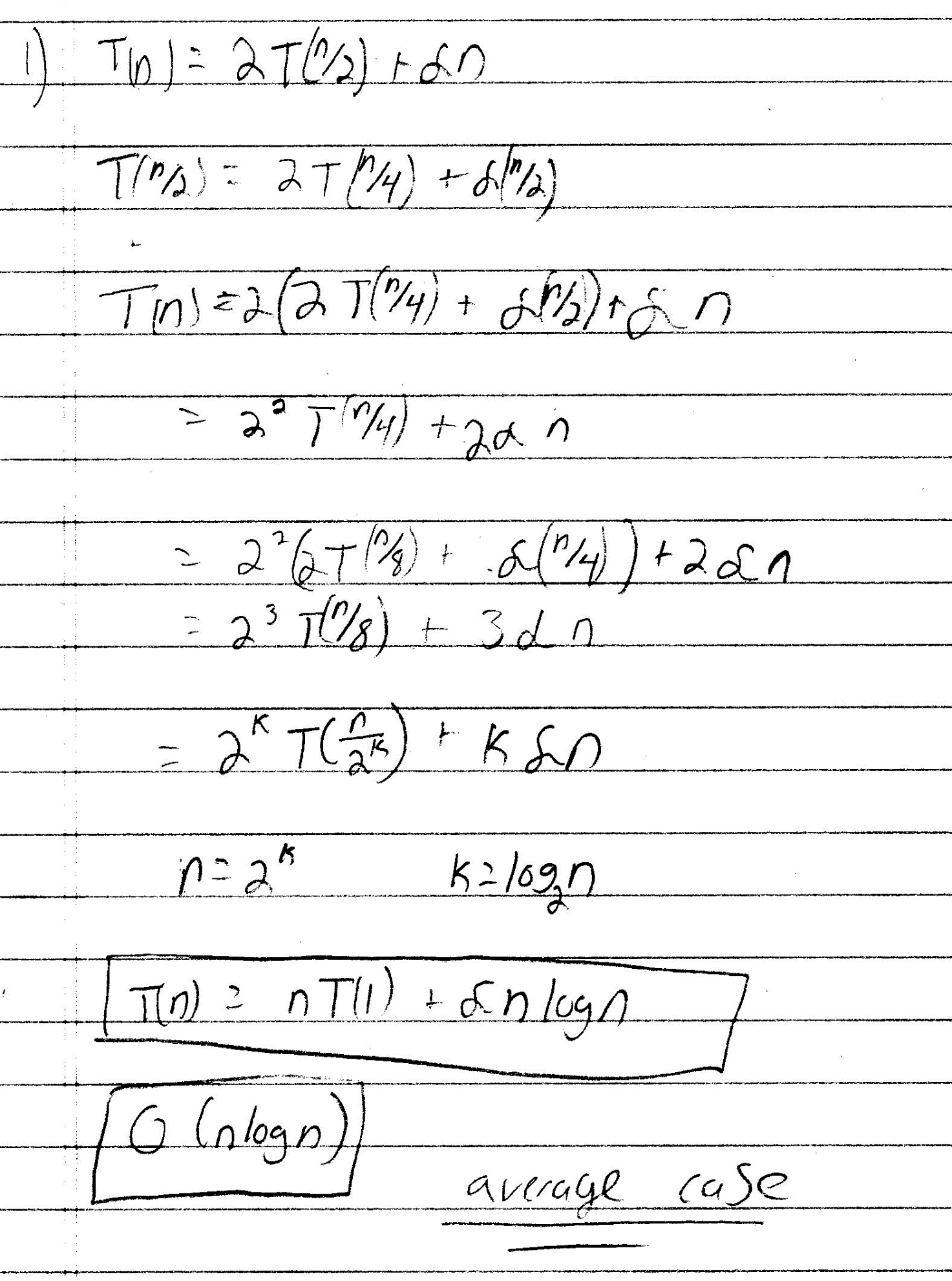
} while (up < down); // Repeat while up is left of down.

swap(arr[first],arr[down]);

return down;

}





1. Programmatically generate three type of arrays of size 1000: an array that is sorted in ascending order, a reversed array (an array that is sorted in descending order), and a random array.

* Programmatically sort the three arrays using insertion sort, merge sort, quick sort, heap sort, counting sort and radix sort. For each sorting algorithm, calculate the number of the number of comparisons (see below).
* Discuss whether your results match the big O of these algorithms.
* **Note:** You can implement the code yourself, or you can get it from an external resource. If you get it from an external resource, cite that resource please.
* **Submission:** Please submit the source code as well the table that summarize the results: This is what the table should look like:

**Comparing the number of comparisons**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Random Array** | **Sorted Array** | **Reversed Array** |
| Insertion Sort | 247,010 | 0 | 499,500 |
| Merge Sort | 10975 | 10975 | 10975 |
| Heap Sort | 19,767 | 21,672 | 17,657 |
| Quick Sort | 6,165 | 500,499 | 250,499 |
| Counting Sort | 0 | 0 | 0 |
| Radix Sort | 4 | 999 | 0 |

**Note: For non-comparison algorithms, count the number of iterations in the loops**

|  |  |  |
| --- | --- | --- |
|  | **Sorted Array O()** | **Reversed Array O()** |
| Insertion Sort | O(n) | O(n2) |
| Merge Sort | O(nlogn) | O(nlogn) |
| Heap Sort | O(nlogn) | O(nlogn) |
| Quick Sort | O(n2) | O(nlogn) |
| Counting Sort | O(n+k) | O(n+k) |
| Radix Sort | O(nk) | O(nk) |

**Insertion sort get exponentially bigger as the array gets more reversed, and that matches up exactly.**

**Merge Sort stays the same since it checks almost half of everything and all are very similar.**

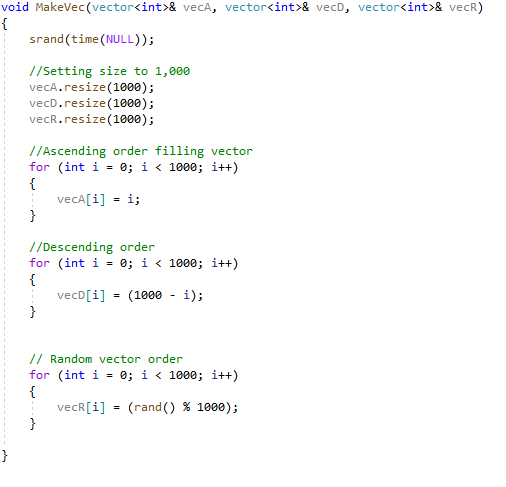
**Heap sort each case is similar in all cases since it uses heapify function which is still O(nlogn)**

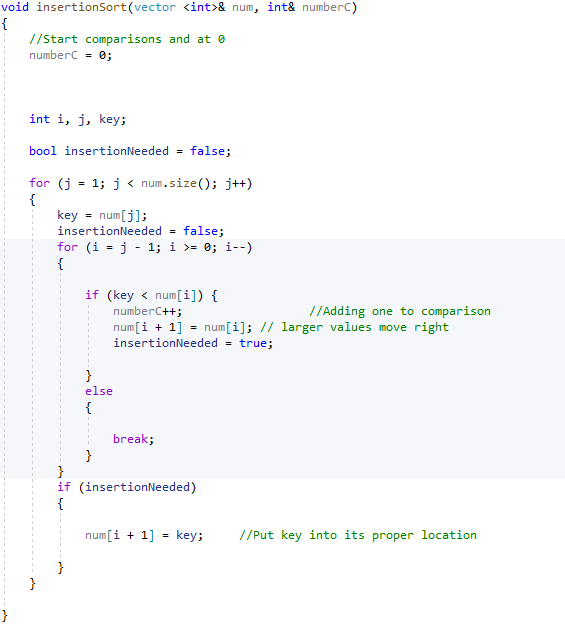
**Quick Sort gets worse exponentially as the array is sorted. Since the comparison in the code is from least to greatest. The counting of each comparison is worse with is sorted, and half that when it is reversed.**

**Counting Sort doesn’t do any comparisons so if came to a count 0.**

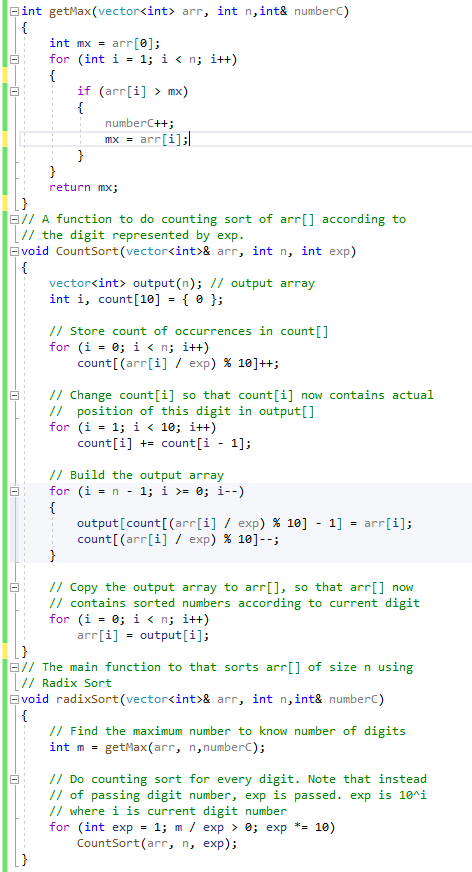
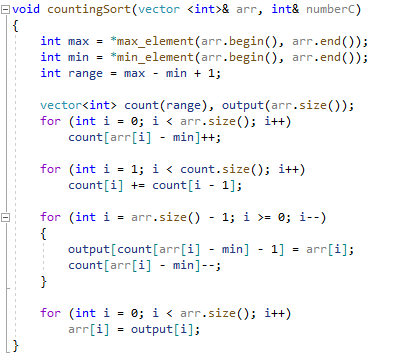
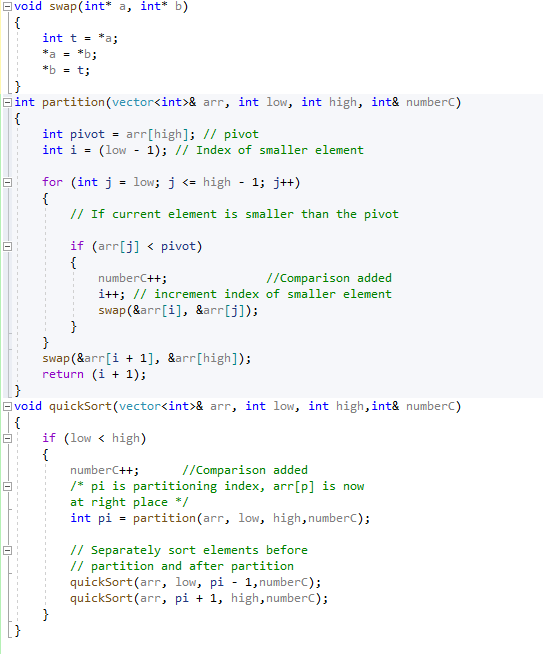
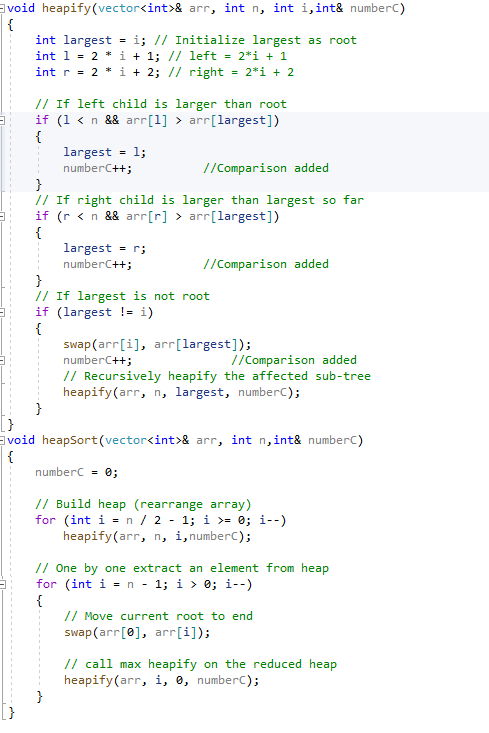
**Radix sort don’t compare, but does compare when needing max value. So, I counted those, and it still came up small count number.**

**Here is some of the code in C++**

****

****

****

****