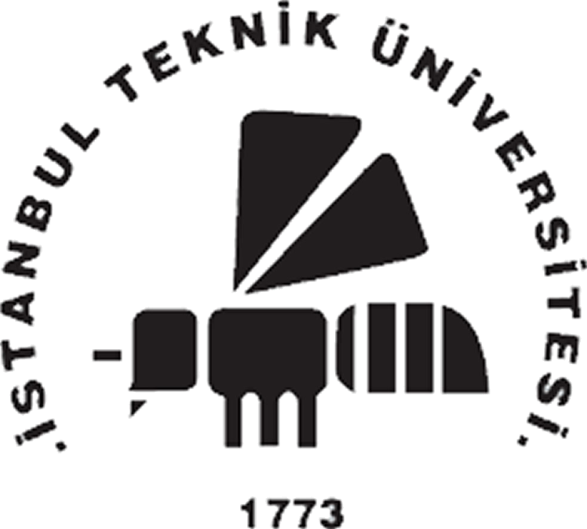
**I.T.U.**

**Faculty of Electric-Electronic**

**Computer Engineering**



Lesson name: Advanced Data Structures

Lesson Code: BLG 381E

Name Surname: Abdullah AYDEĞER

Number: 040090533

Instructor’s Name: Zehra ÇATALTEPE

Due Date: 29.11.2011

**What This Report Includes?**

* **Introduction**
* **Structure**
* **Functions**
* **Running Times**
* **Memory Spent**

**Introduction**

I’ve used Microsoft Visual Studio compiler to compile my codes. I wrote one header which includes structure for implement all necessary functions with more objective way. In addition, two cpp files, one file for main function and the other file for implement structure’s functions.

In main program, I’ve read information from input files, and later I’ve called sorting functions with for loop. Finally of the for loop, I’ve calculated running time in ms for each sorting functions.

I’ve used input files and write sorted output files for heap sort too. So I can compare heap sort algorithm to others very easily.

**Structure**

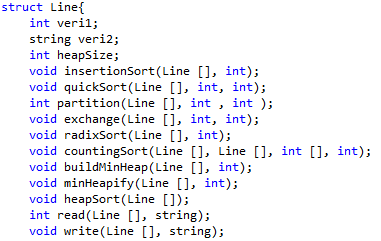
****

Figure 1. Structure

This structure is written for implement all necessary functions and necessary variables in one header.

**Functions**

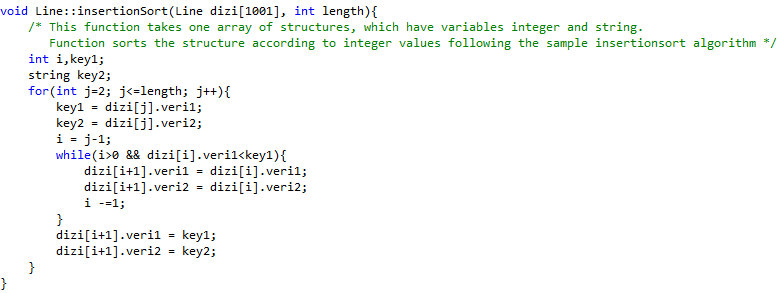
****

Figure 2. Insertion Sort

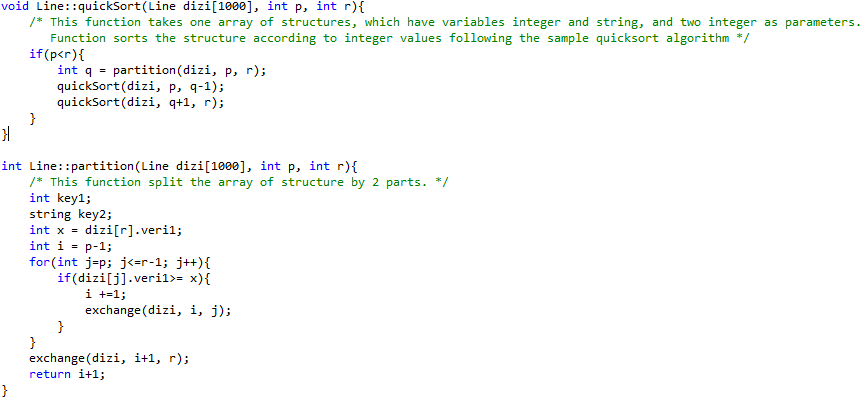


Figure 3. Quick Sort

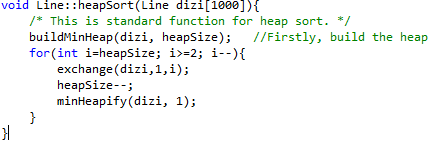


Figure 4. Heap Sort

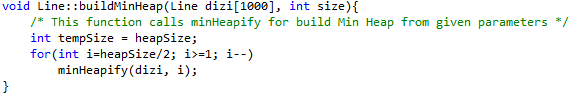


Figure 5. Build Min Heap

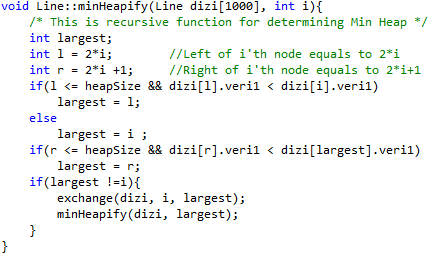
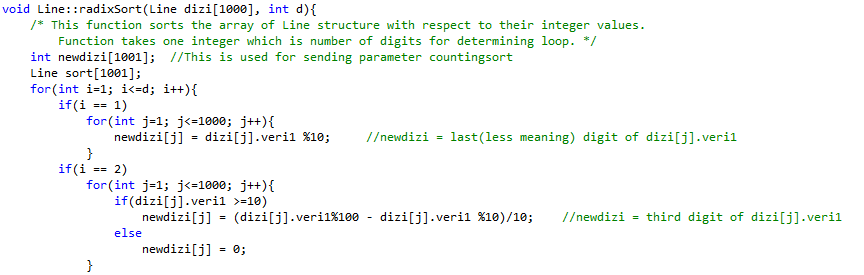


Figure 6. Min Heapify



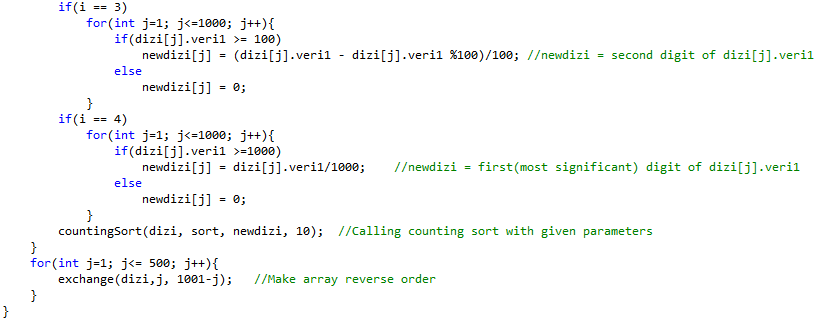


Figure 7. Radix Sort

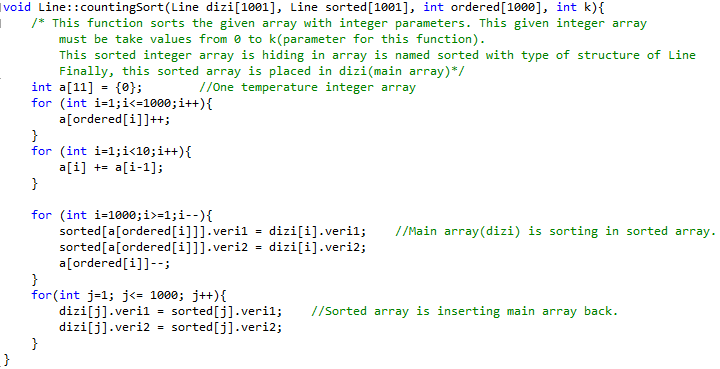


Figure 8. Counting Sort

All necessary sorting functions are given above with necessary explanation in the photos.

**Running Times**

Running times of sorting functions with given array of structures parameters are tabled below. Here I’ve called all sorting functions for 100 times. If you want to make it for 1000 times, only change the LOOP constant to 1000 in main.cpp file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Files | InsertionSort | QuickSort | RadixSort | HeapSort |
| Data1.txt | 25608 ms | 84432 ms | 3867 ms | 4420 ms |
| Data2.txt | 13845 ms | 3600 ms | 3915 ms | 4753 ms |
| Data3.txt | 1451 ms | 169182 ms | 3916 ms | 4994 ms |

**Memory Spent**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Files | InsertionSort | QuickSort | RadixSort | HeapSort |
| Data1.txt | (4+32)\*999 | 44\*500 | (40+4)\*1001+4\*11 | >40\*1000 |
| Data2.txt | (4+32)\*999 | 44\*500 | (40+4)\*1001+4\*11 | =40\*1000 |
| Data3.txt | (4+32)\*999 | 44\*500 | (40+4)\*1001+4\*11 | <40\*1000 |

Unities of memory spent are bytes.

Conclusion, by seeing running times; we can see easily that for all input files the most stable algorithm is radix sort. But if we know that our input is randomized, than quicksort algorithm sorts the inputs fastest. In addition, insertionsort algorithm sorts the sorted input rapidly. Heapsort algorithm is stable sort like radixsort but it’s slower.

According to memory spent radix sort is very expensive sort. Insertion sort spends same memory for all inputs. In quicksort we cannot calculate the memory spent for randomized input, but it’s nearly as seen. Furthermore, heapsort spends a little bit memory while exchanging two elements of array(here is structure array).