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

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

```
struct Line{
    int veri1;
    string veri2;
    int heapSize;
    void insertionSort(Line [], int);
    void quickSort(Line [], int, int);
    int partition(Line [], int , int );
    void exchange(Line [], int, int);
    void radixSort(Line [], int);
    void countingSort(Line [], Line [], int [], int);
    void buildMinHeap(Line [], int);
    void minHeapify(Line [], int);
    void heapSort(Line []);
    int read(Line [], string);
    void write(Line [], string);
```

Figure 1. Structure

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

Functions

```
void Line::insertionSort(Line dizi[1001], int length){
    /* This function takes one array of structures, which have variables integer and string.
       Function sorts the structure according to integer values following the sample insertionsort algorithm */
    int i,key1;
    string key2;
    for(int j=2; j<=length; j++){
        key1 = dizi[j].veri1;
        key2 = dizi[j].veri2;
        i = j-1;
        while(i>0 && dizi[i].veri1<key1){
            dizi[i+1].veri1 = dizi[i].veri1;
            dizi[i+1].veri2 = dizi[i].veri2;
            i -=1;
        }
        dizi[i+1].veri1 = key1;
        dizi[i+1].veri2 = key2;
    }
}
```

Figure 2. Insertion Sort

```
void Line::quickSort(Line dizi[1000], int p, int r){
    /* This function takes one array of structures, which have variables integer and string, and two integer as parameters.
       Function sorts the structure according to integer values following the sample quicksort algorithm */
    if(p<r){
        int q = partition(dizi, p, r);
        quickSort(dizi, p, q-1);
        quickSort(dizi, q+1, r);
    }
}

int Line::partition(Line dizi[1000], int p, int r){
    /* This function split the array of structure by 2 parts. */
    int key1;
    string key2;
    int x = dizi[r].veri1;
    int i = p-1;
    for(int j=p; j<=r-1; j++){
        if(dizi[j].veri1>= x){
            i +=1;
            exchange(dizi, i, j);
        }
    }
    exchange(dizi, i+1, r);
    return i+1;
}
```

Figure 3. Quick Sort

```
void Line::heapSort(Line dizi[1000]){
    /* This is standard function for heap sort. */
    buildMinHeap(dizi, heapSize);   //Firstly, build the heap
    for(int i=heapSize; i>=2; i--){
        exchange(dizi,1,i);
        heapSize--;
        minHeapify(dizi, 1);
    }
}
```

Figure 4. Heap Sort

```
void Line::buildMinHeap(Line dizi[1000], int size){
    /* This function calls minHeapify for build Min Heap from given parameters */
    int tempSize = heapSize;
    for(int i=heapSize/2; i>=1; i--)
        minHeapify(dizi, i);
}
```

Figure 5. Build Min Heap

```
void Line::minHeapify(Line dizi[1000], int i){
    /* This is recursive function for determining Min Heap */
    int largest;
    int l = 2*i;        //Left of i'th node equals to 2*i
    int r = 2*i +1;     //Right of i'th node equals to 2*i+1
    if(l <= heapSize && dizi[l].veri1 < dizi[i].veri1)
        largest = l;
    else
        largest = i ;
    if(r <= heapSize && dizi[r].veri1 < dizi[largest].veri1)
        largest = r;
    if(largest !=i){
        exchange(dizi, i, largest);
        minHeapify(dizi, largest);
    }
}
```

Figure 6. Min Heapify

```
void Line::radixSort(Line dizi[1000], int d){
   /* This function sorts the array of Line structure with respect to their integer values.
       Function takes one integer which is number of digits for determining loop. */
   int newdizi[1001];  //This is used for sending parameter countingsort
   Line sort[1001];
   for(int i=1; i<=d; i++){
       if(i == 1)
           for(int j=1; j<=1000; j++){
               newdizi[j] = dizi[j].veri1 %10;     //newdizi = last(less meaning) digit of dizi[j].veri1
           }
       if(i == 2)
           for(int j=1; j<=1000; j++){
               if(dizi[j].veri1 >=10)
                   newdizi[j] = (dizi[j].veri1%100 - dizi[j].veri1 %10)/10;    //newdizi = third digit of dizi[j].veri1
               else
                   newdizi[j] = 0;
           }

       if(i == 3)
           for(int j=1; j<=1000; j++){
               if(dizi[j].veri1 >= 100)
                   newdizi[j] = (dizi[j].veri1 - dizi[j].veri1 %100)/100; //newdizi = second digit of dizi[j].veri1
               else
                   newdizi[j] = 0;
           }
       if(i == 4)
           for(int j=1; j<=1000; j++){
               if(dizi[j].veri1 >=1000)
                   newdizi[j] = dizi[j].veri1/1000;    //newdizi = first(most significant) digit of dizi[j].veri1
               else
                   newdizi[j] = 0;
           }
       countingSort(dizi, sort, newdizi, 10);  //Calling counting sort with given parameters
   }
   for(int j=1; j<= 500; j++){
       exchange(dizi,j, 1001-j);   //Make array reverse order
   }
}
```

Figure 7. Radix Sort

```
void Line::countingSort(Line dizi[1001], Line sorted[1001], int ordered[1000], int k){
    /* This function sorts the given array with integer parameters. This given integer array
        must be take values from 0 to k(parameter for this function).
        This sorted integer array is hiding in array is named sorted with type of structure of Line
        Finally, this sorted array is placed in dizi(main array)*/
    int a[11] = {0};        //One temperature integer array
    for (int i=1;i<=1000;i++){
        a[ordered[i]]++;
    }
    for (int i=1;i<10;i++){
        a[i] += a[i-1];
    }

    for (int i=1000;i>=1;i--){
        sorted[a[ordered[i]]].veri1 = dizi[i].veri1;    //Main array(dizi) is sorting in sorted array.
        sorted[a[ordered[i]]].veri2 = dizi[i].veri2;
        a[ordered[i]]--;
    }
    for(int j=1; j<= 1000; j++){
        dizi[j].veri1 = sorted[j].veri1;    //Sorted array is inserting main array back.
        dizi[j].veri2 = sorted[j].veri2;
    }
}
```

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).