# RAJSHAHI UNIVERSITY OF ENGINEERING AND TECHNOLOGY


**Lab report: 01**

**Date of Experiment: 29.10.2018**
**Date of Submission: 05.11.2018**

**Submitted to:**
Biprodip Pal
Assistant Professor,
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

**Submitted by:**
Riyad Morshed Shoeb
Roll No: 1603013
Section: A
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

**Problem-A: Efficiency consideration in terms of computation time for Bubble sort and Merge sort.**

**Bubble Sort**

**Code:**

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <fstream>
using namespace std;

int main(void)
{
  ifstream read_file("input.txt");
  int input[50000];
  int i, j;
  int temp;
  int n = 50000;
  double start;
  double stop;
  if(read_file.is_open())
  {
    for(i=0; i<n && !read_file.eof(); i++)
    {
      read_file >> input[i];
    }
    read_file.close();
  }
  else
    cout << "Failed to open input file" << endl;
  start = clock();
  for(i=0; i<n; i++)
  {
    for(j=i; j<n; j++)
    {
      if(input[i] > input[j])
      {
        temp = input[i];
        input[i] = input[j];
        input[j] = temp;
      }
    }
  }
  stop = clock();
  for(i=0; i<n; i++)
    cout << input[i] << endl;
  cout << (stop - start);
}
```

**Theoretical Complexity:**

$T(n) = number\ of\ comparison\ in\ every\ iteration$

$= (n-1) + (n-2) + \cdots + 3 + 2 + 1$

$$= 1 + 2 + 3 + \cdots + (n-2) + (n-1) + n - n$$
$$= \frac{n(n-1)}{2} - n$$
$$= \frac{n^2}{2} - \frac{3n}{2}$$
$$= O(n^2)$$

**Merge Sort**

**Code:**

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <algorithm>
using namespace std;

void merge(int input[], int l, int m, int r)
{
  int i, j, k;
  int n1 = m-l+1;
  int n2 = r-m;
  int L[n1];
  int R[n2];
  for (i=0; i<n1; i++)
    L[i] = input[l+i];
  for (j=0; j<n2; j++)
    R[j] = input[m+1+j];
  i = 0;
  j = 0;
  k = l;
  while(i<n1 && j<n2)
  {
    if(L[i] <= R[j])
    {
      input[k] = L[i];
      i++;
    }
    else
    {
      input[k] = R[j];
      j++;
    }
    k++;
  }
  while(i < n1)
  {
    input[k] = L[i];
    i++;
    k++;
  }
  while(j < n2)
  {
    input[k] = R[j];
```

```
    j++;
    k++;
  }
}

void mergeSort(int input[], int l, int r)
{
  int m;
  if(l<r)
  {
    m = (l+r-1)/2;
    mergeSort(input, l, m);
    mergeSort(input, m+1, r);
    merge(input, l, m, r);
  }
}

int main(void)
{
  ifstream read_file("input.txt");
  int input[30000];
  int i, j;
  int n = 30000;
  double start;
  double stop;
  if(read_file.is_open())
  {
    for(i=0; i<n && !read_file.eof(); i++)
    {
      read_file >> input[i];
    }
    read_file.close();
  }
  else
    cout << "Failed to open input file" << endl;
  start = clock();
  mergeSort(input, 0, n-1);
  stop = clock();
  for(i=0; i<n; i++)
    cout << input[i] << endl;
  cout << (stop - start)*100;
}
```

**Theoretical Complexity:**

$T(n) = 2T(n/2) + n$

*here*, $a = b = 2$

*using the master theorem*, $T(n) = O(nlogn)$

**Time Comparison:**

| Number of input | Bubble Sort | Merge sort |
|---|---|---|
| 2000 | 30ms | ~0ms |
| 5000 | 90ms | 9ms |
| 10000 | 273ms | 34ms |
| 20000 | 938ms | 103ms |

| | | |
|---|---|---|
| 30000 | 1894ms | 243ms |
| 40000 | 3189ms | 533ms |
| 50000 | 4866ms | 751ms |

**Discussion:**
Merge sort gives better performance than Bubble sort in terms of time but it is not so efficient in terms of memory since it requires double memory than the memory required by input data.

**Problem-B: Find a better approach than the brute force technique to find the sum of 3 numbers that add up to 0.**

**Using Brute force**

**Code:**

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include <ctime>
using namespace std;

int main(void)
{
  ifstream read_file("input.txt");
  int input[200];
  int i, j, k, n = 200;
  double start, stop;

  if(read_file.is_open())
  {
    for(i=0; i<n && !read_file.eof(); i++)
    {
      read_file >> input[i];
    }
    read_file.close();
  }
  else
    cout << "Failed to open input file" << endl;
  start = clock();
  for(i=0; i<n-2; i++)
    for(j=i+1; j<n-1; j++)
      for(k=j+1; k<n; k++)
        if(input[i] + input[j] + input[k] == 0)
          cout << input[i] << " " << input[j] << " " << input[k] <<
endl;
  stop = clock();
  cout << (stop - start)/1000;
}
```

**Theoretical Complexity:**

$$T(n) = (n-2)[(n-2)+(n-3)+\cdots+2+1]+(n-3)[(n-3)+(n-4)+\cdots+2+1]+\cdots+1*1$$

$$= (n-2)\left[\frac{n(n-1)}{2}-2n+1\right]+(n-3)\left[\frac{n(n-1)}{2}-3n+5\right]+\cdots+1$$

$$= (n-2)\frac{n^2-n-4n+2}{2}+(n-3)\frac{n^2-n-6n+10}{2}+\cdots+1$$

$$\leq (n-2)(n^2-5n+2)+(n-3)(n^2-7n+10)+\cdots+1$$

$$\leq n^3+(n-1)^3+(n-2)^3+(n-3)^3+\cdots+1^3$$

$$= \frac{n(n-1)(2n-1)}{6}$$

$$= \frac{2n^3-3n^2+n}{6}$$

$$= O(n^3)$$

**Using Divide and Conquer**

**Code:**

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <utility>
using namespace std;

pair <bool, int> p;

void merge(int input[], int l, int m, int r)
{
  int i, j, k;
  int n1 = m-l+1;
  int n2 = r-m;
  int L[n1];
  int R[n2];
  for (i=0; i<n1; i++)
    L[i] = input[l+i];
  for (j=0; j<n2; j++)
    R[j] = input[m+1+j];

  i = 0;
  j = 0;
  k = l;
  while(i<n1 && j<n2)
  {
    if(L[i] <= R[j])
    {
      input[k] = L[i];
      i++;
    }
    else
    {
      input[k] = R[j];
      j++;
    }
```

```
    k++;
  }
  while(i < n1)
  {
    input[k] = L[i];
    i++;
    k++;
  }
  while(j < n2)
  {
    input[k] = R[j];
    j++;
    k++;
  }
}

void mergeSort(int input[], int l, int r)
{
  int m;
  if(l<r)
  {
    m = (l+r-1)/2;
    mergeSort(input, l, m);
    mergeSort(input, m+1, r);
    merge(input, l, m, r);
  }
}

void search(int input[], int x, int left, int right)
{
  int mid;
  p = make_pair(false, 0);
  while (left <= right)
  {
    mid = (left+right)/2;
    if(input[mid] == x)
    {
      p = make_pair(true, input[mid]);
      break;
    }
    else if(input[mid] > x)
      right = mid - 1;
    else
      left = mid + 1;
  }
}

int main(void)
{
  ifstream read_file("input.txt");
  int input[2000];
  int i, j;
  int n = 2000;
  double start;
  double stop;
  if(read_file.is_open())
```

```
  {
    for(i=0; i<n && !read_file.eof(); i++)
    {
      read_file >> input[i];
    }
    read_file.close();
  }
  else
    cout << "Failed to open input file" << endl;

  mergeSort(input, 0, n-1);
  start = clock();
  for(i=0; i<n-1; i++)
  {
    for(j=i+1; j<n; j++)
    {
      search(input, -input[i]-input[j], j, n-1);
      if(p.first)
        cout << input[i] << " " << input[j] << " " << p.second <<
endl;
    }
  }
  stop = clock();
  cout << (stop - start)/1000;
}
```

**Theoretical Complexity:**

$T(n) = n^2\left[T\left({}^{n}/_{2}\right) + c\right]$

$from\ master\ theory, we\ can\ say, T\left({}^{n}/_{2}\right) + c = O(logn)$

$\therefore T(n) = O(n^2 logn)$

**Time comparison:**

| Number of Inputs | Iterative method | Divide and conquer |
|---|---|---|
| 100 | 0.074s | 0.186s |
| 200 | 0.666s | 1.143s |
| 500 | 9.732s | 16.246s |
| 1000 | 43.979s | 82.852s |
| 2000 | 333.149s | 172.758s |

**Discussion:**

Divide and Conquer approach gives better performance than Brute force for larger inputs but it is still time costly.