FCE-hw1

Weikun Su

NUID: 001498198

September 2019

Question 1

The codes are shown below. We can change the value of n in line 76. The results are shown in Table 1. The results may different each time you run the code, but the offset can be ignore, it will not influence our analysis.
Analysis: When input size n is less than 38, insertion sort run faster than merge sort. When input size n is larger than 38, merge sort beats insertion sort.

n	insertion sort(μs)	merge sort(μs)
10	0.4	0.7
20	0.9	1.2
30	1.6	1.8
35	2	2.3
38	2.3	2.3
40	2.5	2.4
50	3.6	3

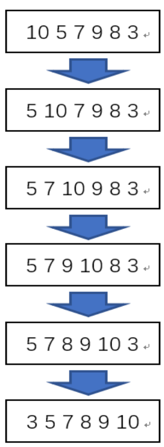
Table 1: Result of the code

```
0  #include <iostream>
   #include <windows.h>
2
   void insertion_sort(int v[], int n)
4  {
       int value;
6      int i, j;
       for (i = 1; i < n; i++)
8      {
           value = v[i];
10         j = i - 1;
           while (j >= 0 && v[j] > value)
12         {
               v[j + 1] = v[j];
14             j--;

16         }
           v[j + 1] = value;
18     }
   }
20
   void merge(int v[], int l, int m, int h)
22 {
       int i, j, k;
```

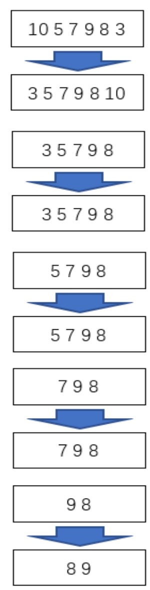
```
    int temp[h - l + 1];
    i = l;
    j = m + 1;
    k = 0;
    while (i <= m && j <= h)
    {
        if (v[i] < v[j])
        {
            temp[k] = v[i];
            i++;
            k++;
        }
        else
        {
            temp[k] = v[j];
            j++;
            k++;
        }
    }
    while (i <= m)
    {
        temp[k] = v[i];
        i++;
        k++;
    }
    while (j <= h)
    {
        temp[k] = v[j];
        j++;
        k++;
    }
    for (i = l; i <= h; i++)
    {
        v[i] = temp[i - l];
    }
}

void merge_sort(int v[], int l, int h)
{
    int m;
    if (l < h)
    {
        m = (l + h) / 2;
        merge_sort(v, l, m);
        merge_sort(v, m+1, h);
        merge(v, l, m, h);
    }
}
int main()
{
    int n;
    int i;
    n = 50; //input size n
    LARGE_INTEGER nFreq;
    LARGE_INTEGER nBeginTime;
    LARGE_INTEGER nEndTime;
    double t_1;
    double t_2;
    int v_1[n];
    int v_2[n];
    for (i = 0; i < n; i++)
    {
        v_1[i] = n - i - 1;
    }
    for (i = 0; i < n; i++)
```

```
    {
        v_2[i] = n - i - 1;
    }
    QueryPerformanceFrequency(&nFreq);
    QueryPerformanceCounter(&nBeginTime); //begin time of insertion sort
    insertion_sort(v_1, n);
    QueryPerformanceCounter(&nEndTime); //end time of insertion sort
    t_1 = (double)(nEndTime.QuadPart-nBeginTime.QuadPart)/(double)nFreq.QuadPart; //
    running time of insertion sort
    QueryPerformanceCounter(&nBeginTime); //begin time of merge sort
    merge_sort(v_2, 0, n-1);
    QueryPerformanceCounter(&nEndTime); //end time of merge sort
    t_2 = (double)(nEndTime.QuadPart-nBeginTime.QuadPart)/(double)nFreq.QuadPart; //
    running time of merge sort
    std::cout << "running time of insertion sort:" << t_1 << "s" << std::endl ;
    std::cout << "running time of merge sort:" << t_2 << "s" << std::endl ;
    return 0;
}
```

Question 2



(a) insertion sort



(b) quick sort

Question 3

True, True, False, False, True

Question 4

1. $T(n) = 8T(n/2) + n$

$$a = 8, b = 2, f(n) = n$$

It belongs to case I:

$$f(n) = O(n^{\log_2 8-\epsilon}) = O(n^{3-\epsilon})$$

let $\epsilon = 2$:

$$f(n) = O(n)$$

Therefore,

$$T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$

2. $T(n) = 8T(n/2) + n^2$

$$a = 8, b = 2, f(n) = n^2$$

It belongs to case I:

$$f(n) = O(n^{\log_2 8-\epsilon}) = O(n^{3-\epsilon})$$

let $\epsilon = 1$:

$$f(n) = O(n^2)$$

Therefore,

$$T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$

3. $T(n) = 8T(n/2) + n^3$

$$a = 8, b = 2, f(n) = n^3$$

It belongs to case II:

$$f(n) = \Theta(n^{\log_2 8} \log_2^k n) = \Theta(n^3 \log_2^k n)$$

let $k = 0$:

$$f(n) = \Theta(n^3)$$

Therefore,

$$T(n) = \Theta(n^{\log_2 8} \log_2 n) = \Theta(n^3 \log_2 n)$$

4. $T(n) = 8T(n/2) + n^4$

$$a = 8, b = 2, f(n) = n^4$$

It belongs to case III:

$$\begin{cases} f(n) = \Omega(n^{\log_2 8+\epsilon}) = \Omega(n^{3+\epsilon}) = \Omega(n^4) & , for \quad \epsilon = 1 \\ \quad 8f(n/2) = \dfrac{n^2}{2} \leq (1-\epsilon')f(n) = \dfrac{1}{2}n^4 & , for \quad \epsilon' = \dfrac{1}{2} \end{cases}$$

Therefore,

$$T(n) = \Theta(f(n)) = \Theta(n^4)$$

Question 5

The recursion tree is shown in the figure with $leaves = 8^{\log_2 n}$. We add all the elements of the tree to get the running time.

$$T(n) = \sum_{i=0}^{\log_2 n} 8^i \frac{n}{2^i} = \sum_{i=0}^{log_2 n} 4^i n$$

This is the summation of a geometric sequence, the result is:

$$T(n) = \frac{4}{3}n^3 - \frac{1}{3}n = \Theta(n^3)$$

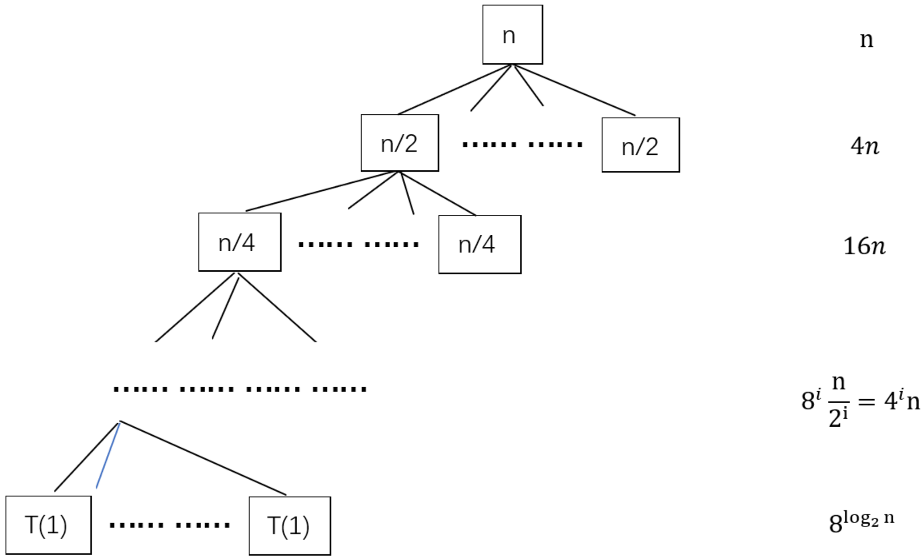


Figure 2: Recursion Tree

Prof 1: $T(n) = \Omega(n^3)$
Assume : $T(k) \leq ck^3$ for $k < n$

$$T(n) = 8T(\frac{n}{2}) + n \geq 8c(\frac{n}{2})^3 + n = cn^3 + n$$

let $c = 1$, $n_0 = 1$, we have $T(n) \geq cn^3$
Therefore, $T(n) = \Omega(n^3)$
prof 2: $T(n) = O(n^3)$
Assume: $T(k) \leq c_1k^3 - c_2k$ for $k < n$

$$T(n) = 8T(\frac{n}{2}) + n \leq 8c_1(\frac{n}{2})^3 - (4c_2 - 1)n = c_1n^3 - c_2n - (3c_2 - 1)n$$

let $c_1 = 1, c_2 = 1, n_0 = 1$, we have $T(n) \leq c_1n^3 - c_2n$
Therefore, $T(n) = O(n^3)$
According to Prof 1 and Prof 2, $T(n)$ is both $O(n^3)$ and $\Omega(n^3)$. So $T(n) = \Theta(n^3)$