

Question 1. Coding: write programs of insertion sort, and mergesort. Find the input size n , that mergesort starts to beat insertion sort in terms of the worst-case running time. You can use `clock_t` function (or other time function for higher precision) to obtain running time. You need to set your input such that it results in the worst-case running time. Report running time of each algorithm for each input size n .

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
using namespace std;

// The clock_t function is not applicable because it is only accurate to the millisecond level.

void Merge(int origin[], int opera[], int outset, int mid, int end){ // Merge process of merge sort.
    int i = outset, j = mid + 1, k = outset;
    while(i != mid + 1 && j != end + 1){
        if(origin[i] > origin[j]){
            opera[k] = origin[j];
            k = k + 1;
            j = j + 1;
        }
        else{
            opera[k] = origin[i];
            k = k + 1;
            i = i + 1;
        }
    }
    while(i != mid + 1){
        opera[k] = origin[i];
        k = k + 1;
        i = i + 1;
    }
    while(j != end + 1){
        opera[k] = origin[j];
        k = k + 1;
        j = j + 1;
    }
    for(i = outset; i <= end; i++)
        origin[i] = opera[i];
}
```

```

void Merge_Sort(int origin[], int opera[], int outset, int end){ // Partition process of merge sort.
    int mid;
    if(outset < end){
        mid = outset + (end - outset) / 2;
        Merge_Sort(origin, opera, outset, mid);
        Merge_Sort(origin, opera, mid + 1, end);
        Merge(origin, opera, outset, mid, end);
    }
}

void Insertion_Sort(int v[], int n){ // Cite from the example of the assignment.
    int value;
    int i, j;
    for(i = 1; i < n; i++){
        value = v[i];
        j = i - 1;
        while( j >= 0 && v[j] > value){
            v[j + 1] = v[j];
            j--;
        }
        v[j + 1] = value;
    }
}

int main(){
    int test[1000];
    test[0] = 999; // Create array test[] like [999, 998] or [999, 998, 997], etc.
                    // Create a worst-case scenario for each n-scale (n ∈ [1,1000]) array.
    for(int k = 0; k < 999 ; k++){
        test[k + 1] = test[k] - 1;

        int count = 1;
        while(true){
            int test2[count + 1], test_insertion[count + 1], test_merge[count + 1];
            for(int p = 0; p <= count; p++){
                test2[p] = test[p];
                test_insertion[p] = test[p];
                test_merge[p] = test[p];
            }
        }
    }
}

```

```

    }

    LARGE_INTEGER initial_time;
    double cpu_run_time = 0;
    LARGE_INTEGER end_time;
    double f2;
    LARGE_INTEGER f;
    QueryPerformanceFrequency(&f); // Cite from Microsoft C++ documents
    . To count the frequency of my computer's cpu.
    f2 = (double)f.QuadPart;

    QueryPerformanceCounter(&initial_time); // first time stamp.
    Insertion_Sort(test_insertion, count + 1);
    QueryPerformanceCounter(&end_time); // Last time stamp.
    cpu_run_time = ((double)end_time.QuadPart - (double)initial_time.Qu
adPart) / f2;
    printf("insertion_sort's cpu runtime: %fs ",cpu_run_time);

    LARGE_INTEGER initial_time2;
    double cpu_run_time2 = 0;
    LARGE_INTEGER end_time2;
    double f2_2;
    LARGE_INTEGER fr2;
    QueryPerformanceFrequency(&fr2);
    f2_2 = (double)fr2.QuadPart;

    QueryPerformanceCounter(&initial_time2); // first time stamp.
    int b[count + 1];
    Merge_Sort(test_merge, b, 0, count);
    QueryPerformanceCounter(&end_time2); // Last time stamp.
    cpu_run_time2 = ((double)end_time2.QuadPart - (double)initial_time2
.QuadPart) / f2_2;
    printf("merge_sort's cpu runtime: %fs\n",cpu_run_time2);

    if(cpu_run_time2 < cpu_run_time)break; /* When the runtime of merg
e sort is less than insertion sort,
                                                jump out of the loop to
get the critical value n. */
    else count++; // if not, continue.
    }
    std::cout << "n that Merge_Sort beats Insertion_Sort: " << count <<
endl;
    return 0;
}

```

Results:

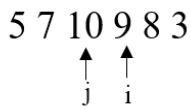
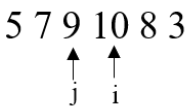
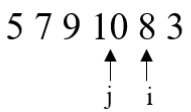
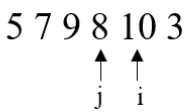
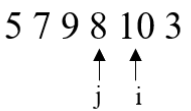
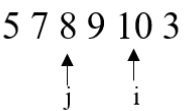
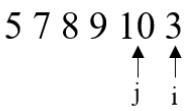
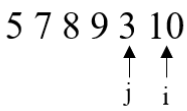
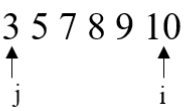
[illegible]

Question 2. You are given with an array $\{10, 5, 7, 9, 8, 3\}$. Show the arrangement of the array for each iteration during insertion sort. You are given with the same array. Show the arrangement of the array for each iteration of the Partition subroutine of quicksort and the result of Partition subroutine.

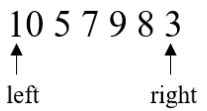
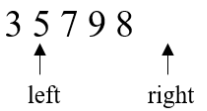
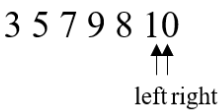
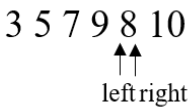
Insertion sort:

1. Initial array: 10 5 7 9 8 3. j points to 10, i points to 5. Since $10 > 5$, exchange i and j . Resulting array: 5 10 7 9 8 3.

2. Array: 5 10 7 9 8 3. j points to 10, i points to 7. Since $10 > 7$, exchange i and j . Resulting array: 5 7 10 9 8 3.

3.  exchange i and j 
4.  exchange i and j 
5.  $a[j-1] > a[j]$ 
6.  exchange i and j 
7. Repeat and finally get 

Quicksort(always select the first element of the array as the pivot):

1.  10 is pivot and $a[\text{right}] < \text{pivot}$ 
2. 5, 7, 9, 8 are all smaller than pivot 
3. Now consider the subarray {3, 5, 7, 9, 8}, 3 is pivot but evidently 3 is the smallest one. Then 5 and 7 become pivot one after another, they are all smaller than numbers to their right.
4. So in the last iteration 9 is pivot, comparing with 8 and finally make an exchange.
-  and the final array is {3, 5, 7, 8, 9, 10}.

Question 3. True or False

$n + 3 \in \Omega(n)$: True
 $n + 3 \in O(n^2)$: True
 $n + 3 \in \Theta(n^2)$: False
 $2^{n+1} \in O(n + 1)$: False
 $2^{n+1} \in \Theta(2^n)$: True

Question 4. Using the master method, determine $T(n)$ for the following recurrences.

$$T(n) = 8T\left(\frac{n}{2}\right) + n$$

From the formula as above, we can get $a = 8$, $b = 2$ and $f(n) = n$. Because $f(n) = n < n^{\log_b a} = n^3$, $T(n) \in \Theta(n^3)$.

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

From the formula as above, we can get $a = 8$, $b = 2$ and $f(n) = n^2$. Because $f(n) = n^2 < n^{\log_b a} = n^3$, $T(n) \in \Theta(n^3)$.

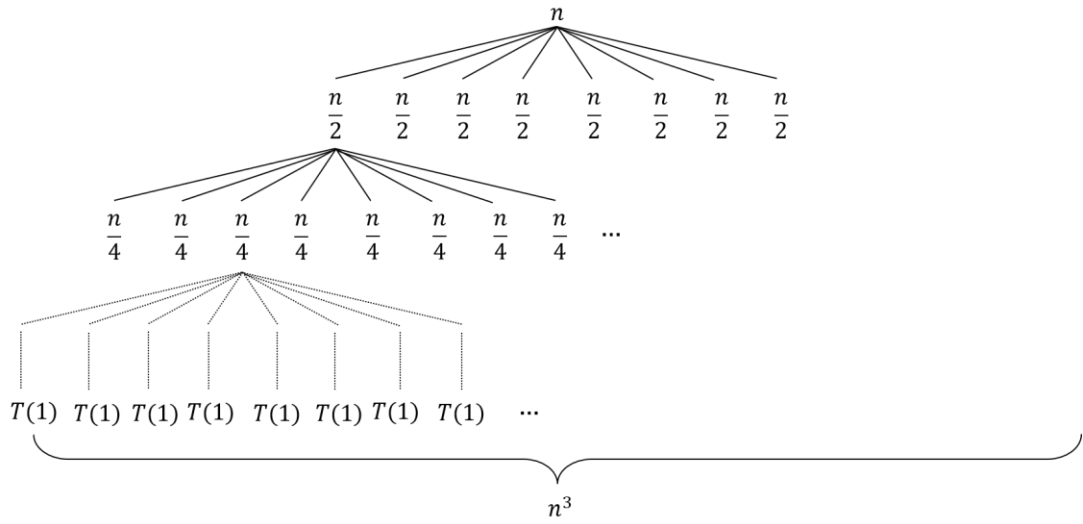
$$T(n) = 8T\left(\frac{n}{2}\right) + n^3$$

From the formula as above, we can get $a = 8$, $b = 2$ and $f(n) = n^3$. Because $f(n) = n^3 = n^{\log_b a} = n^3$, $T(n) \in \Theta(n^3 \log n)$.

$$T(n) = 8T\left(\frac{n}{2}\right) + n^4$$

From the formula as above, we can get $a = 8$, $b = 2$ and $f(n) = n^4$. Because $f(n) = n^4 > n^{\log_b a} = n^3$, $T(n) \in \Theta(n^4)$.

Question 5. Draw recursion tree for $T(n) = 8T\left(\frac{n}{2}\right) + n$. And prove the obtained $T(n)$ by substitution method.



Assume that $T\left(\frac{n}{2}\right) \in O(n^3)$, and there exist positive c and N when $n > N$,

$T\left(\frac{n}{2}\right) < cn^3$. So according to $T(n) = 8T\left(\frac{n}{2}\right) + n$, $T(n) < 8cn^3 + n \in O(n^3)$.

Assume that $T\left(\frac{n}{2}\right) \in \Omega(n^3)$, and there exist positive c and N when $n > N$,

$T\left(\frac{n}{2}\right) > cn^3$. So according to $T(n) = 8T\left(\frac{n}{2}\right) + n$, $T(n) > 8cn^3 + n \in \Omega(n^3)$.

As mentioned above, $T\left(\frac{n}{2}\right) \in \Theta(n^3)$.