HW1_YIXIAO CHEN_002198256

Q1:

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.*

[Q1_my c++ code]

```
#include <iostream>
#include <climits>
#include <ratio>
#include <chrono>
using namespace std;
using namespace std::chrono;
void insertion_sort(int v[], int n)
{
	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;
	}
}
void merge(int a[], int p, int mid, int r)
```

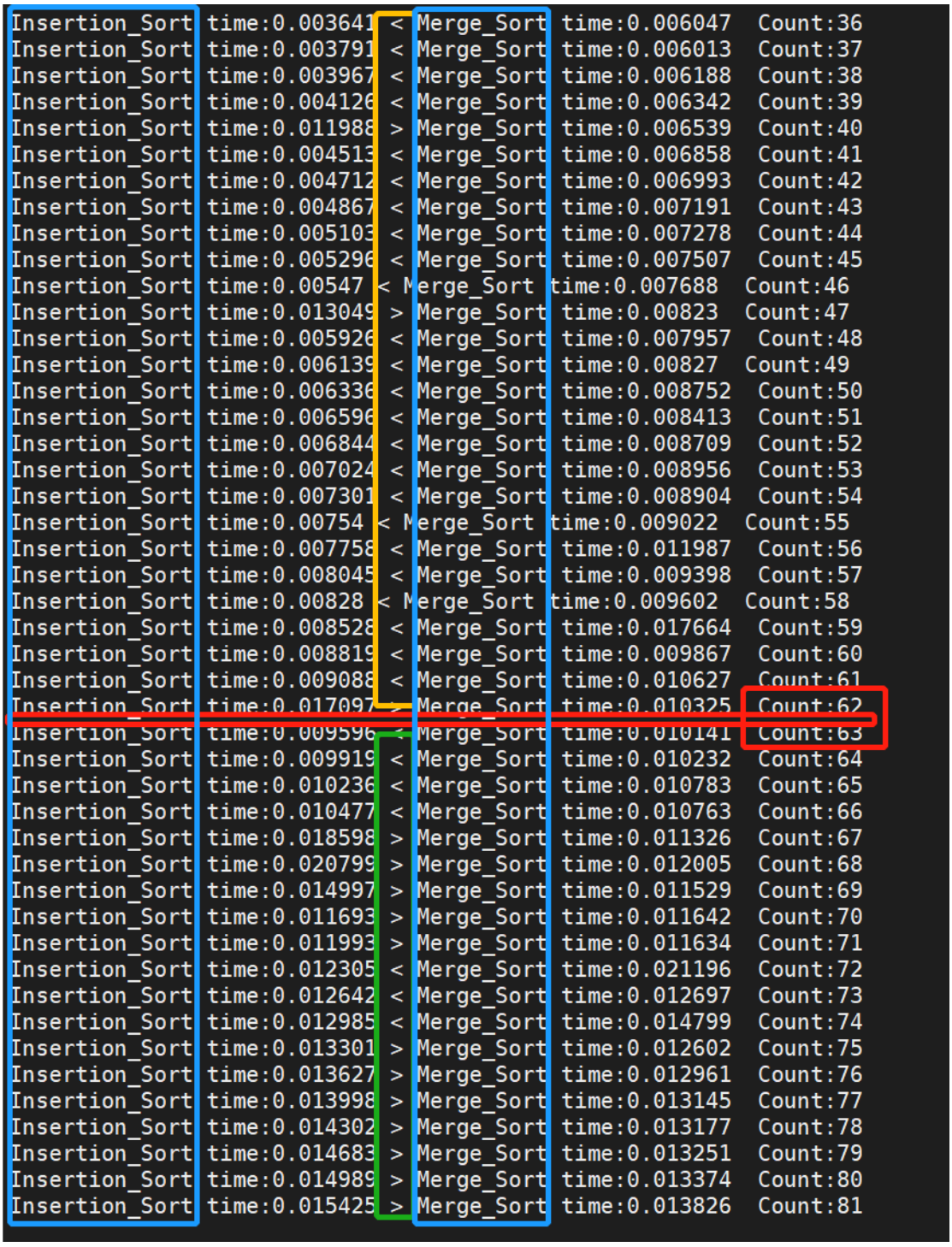
```
{
    int n1 = mid - p + 1;
    int n2 = r - mid;
    int* L = new int[n1 + 2];
    int* R = new int[n2 + 2];
    int i, j;
    for (i = 1; i <= n1; i++)
        L[i] = a[p + i - 1];
    for (j = 1; j <= n2; j++)
        R[j] = a[mid + j];
    i = 1;
    j = 1;
    L[n1 + 1] = INT_MAX;
    R[n2 + 1] = INT_MAX;
    int k;
    for (k = p; k <= r; k++)
    {
        if (L[i] <= R[j])
        {
            a[k] = L[i];
            i = i + 1;
        }
        else
        {
            a[k] = R[j];
            j = j + 1;
        }
    }
    delete[]L;
```

```
	delete[]R;
}
void merge_sort(int a[], int p, int r)
{
	if (p < r)
	{
		int mid = (p + r) / 2;
		merge_sort(a, p, mid);
		merge_sort(a, mid + 1, r);
		merge(a, p, mid, r);
	}
}
void print(int a[], int n)
{
	int i;
	for (i = 0; i < n; i++)
		cout << a[i] << " ";
	cout << endl;
}
int main()
{

	cout << "time competition:[insertion_sort/merge_sort]" << endl;
	for (int i = 1; i <= 80; i++)
	{
		int* v, * a, * o;
		v = new int[i];
		a = new int[i];
		o = new int[i];
```

```
        for (int j = 0; j < i; j++)
        {
            v[j] = i-j;
            a[j] = v[j];
            o[j] = v[j];
        }
        high_resolution_clock::time_point t1 = high_resolution_clock::now();
        insertion_sort(v, i);
        high_resolution_clock::time_point t2 = high_resolution_clock::now();
        duration<double, std::ratio<1, 1000>> duration_IS =
duration_cast<duration<double, std::ratio<1, 1000>>>(t2 - t1);
        high_resolution_clock::time_point t3 = high_resolution_clock::now();
        merge_sort(a, 0, i - 1);
        high_resolution_clock::time_point t4 = high_resolution_clock::now();
        duration<double, std::ratio<1, 1000>> duration_MS =
duration_cast<duration<double, std::ratio<1, 1000>>>(t4 - t3);
        if (duration_IS > duration_MS)
        {
            cout << "\nInsertion_Sort time:" << duration_IS.count() << " >
Merge_Sort time:" << duration_MS.count() << "    Count:" << i << endl;
            cout << "origional_array: " << "";
            print(o, i);
            cout << "Insertion_Sort : " << "";
            print(v, i);
            cout << "    Merge_Sort      : " << "";
            print(a, i);
            cout<< endl;
        }
        else if (duration_IS < duration_MS)
        {
```

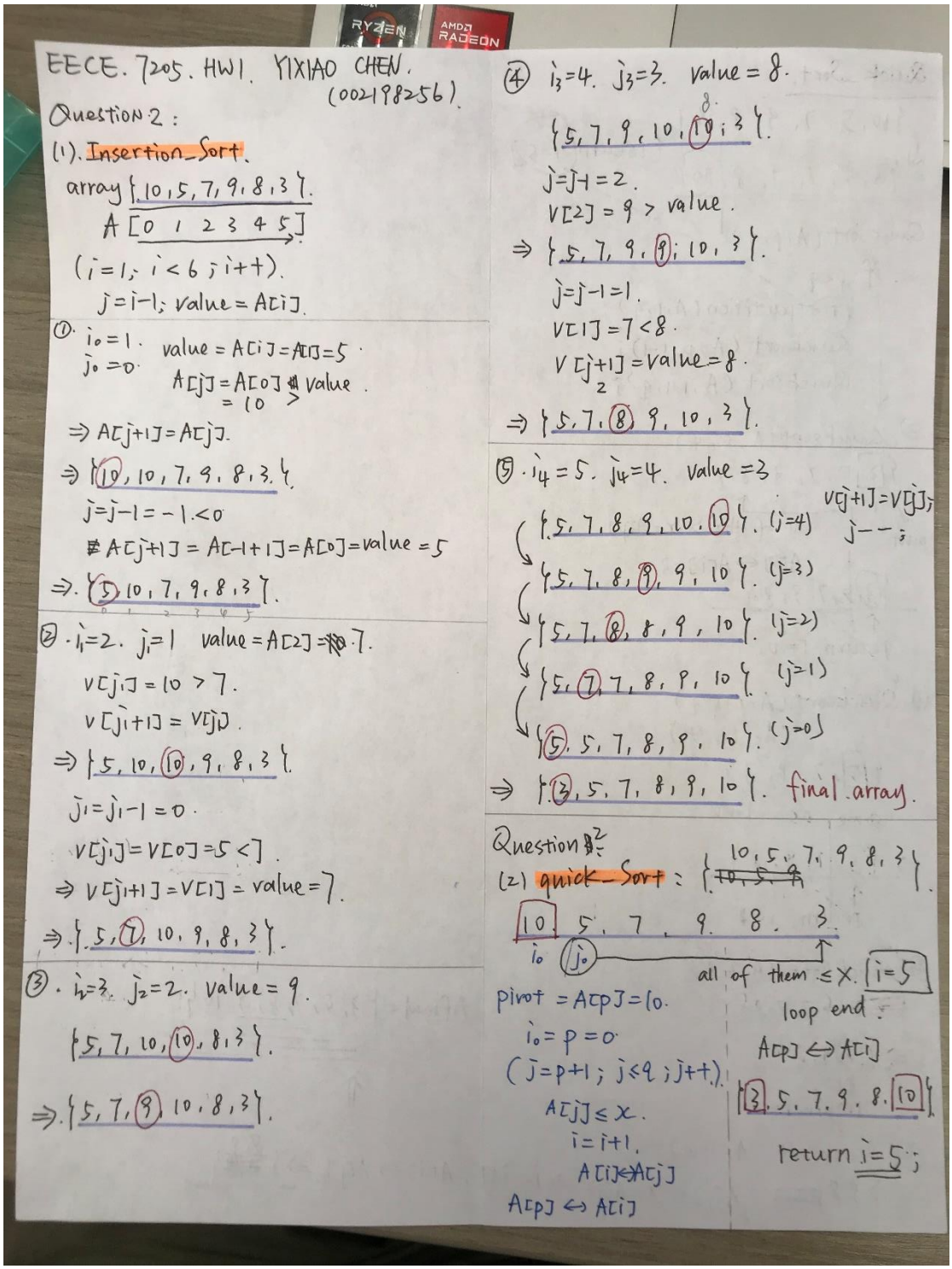
```
            cout << "\nInsertion_Sort time:" << duration_IS.count() << " < Merge_Sort time:" << duration_MS.count() << "    Count:" << i << endl;
            cout << "origional_array: " << "";
            print(o, i);
            cout << "Insertion_Sort : " << "";
            print(v, i);
            cout << "    Merge_Sort      : " << "";
            print(a, i);
            cout << endl;
        }
        else if (duration_IS == duration_MS)
        {
            cout << "\nInsertion_Sort time:" << duration_IS.count() << " = Merge_Sort time:" << duration_MS.count() << "    Count:" << i << endl;
            cout << "origional_array: " << "";
            print(o, i);
            cout << "Insertion_Sort : " << "";
            print(v, i);
            cout << "    Merge_Sort      : " << "";
            print(a, i);
            cout << endl;
        }
        delete[]v;
    }

}
```

[Q1_my result:]


Insertion_Sort time:0.003641 < Merge_Sort time:0.006047 Count:36
Insertion_Sort time:0.003791 < Merge_Sort time:0.006013 Count:37
Insertion_Sort time:0.003967 < Merge_Sort time:0.006188 Count:38
Insertion_Sort time:0.004126 < Merge_Sort time:0.006342 Count:39
Insertion_Sort time:0.011988 > Merge_Sort time:0.006539 Count:40
Insertion_Sort time:0.004513 < Merge_Sort time:0.006858 Count:41
Insertion_Sort time:0.004712 < Merge_Sort time:0.006993 Count:42
Insertion_Sort time:0.004867 < Merge_Sort time:0.007191 Count:43
Insertion_Sort time:0.005103 < Merge_Sort time:0.007278 Count:44
Insertion_Sort time:0.005296 < Merge_Sort time:0.007507 Count:45
Insertion_Sort time:0.00547 < Merge_Sort time:0.007688 Count:46
Insertion_Sort time:0.013049 > Merge_Sort time:0.00823 Count:47
Insertion_Sort time:0.005926 < Merge_Sort time:0.007957 Count:48
Insertion_Sort time:0.006139 < Merge_Sort time:0.00827 Count:49
Insertion_Sort time:0.006336 < Merge_Sort time:0.008752 Count:50
Insertion_Sort time:0.006596 < Merge_Sort time:0.008413 Count:51
Insertion_Sort time:0.006844 < Merge_Sort time:0.008709 Count:52
Insertion_Sort time:0.007024 < Merge_Sort time:0.008956 Count:53
Insertion_Sort time:0.007301 < Merge_Sort time:0.008904 Count:54
Insertion_Sort time:0.00754 < Merge_Sort time:0.009022 Count:55
Insertion_Sort time:0.007758 < Merge_Sort time:0.011987 Count:56
Insertion_Sort time:0.008045 < Merge_Sort time:0.009398 Count:57
Insertion_Sort time:0.00828 < Merge_Sort time:0.009602 Count:58
Insertion_Sort time:0.008528 < Merge_Sort time:0.017664 Count:59
Insertion_Sort time:0.008819 < Merge_Sort time:0.009867 Count:60
Insertion_Sort time:0.009088 < Merge_Sort time:0.010627 Count:61
Insertion_Sort time:0.017097 > Merge_Sort time:0.010325 Count:62
Insertion_Sort time:0.009596 < Merge_Sort time:0.010141 Count:63
Insertion_Sort time:0.009919 < Merge_Sort time:0.010232 Count:64
Insertion_Sort time:0.010236 < Merge_Sort time:0.010783 Count:65
Insertion_Sort time:0.010477 < Merge_Sort time:0.010763 Count:66
Insertion_Sort time:0.018598 > Merge_Sort time:0.011326 Count:67
Insertion_Sort time:0.020799 > Merge_Sort time:0.012005 Count:68
Insertion_Sort time:0.014997 > Merge_Sort time:0.011529 Count:69
Insertion_Sort time:0.011693 > Merge_Sort time:0.011642 Count:70
Insertion_Sort time:0.011993 > Merge_Sort time:0.011634 Count:71
Insertion_Sort time:0.012305 < Merge_Sort time:0.021196 Count:72
Insertion_Sort time:0.012642 < Merge_Sort time:0.012697 Count:73
Insertion_Sort time:0.012985 < Merge_Sort time:0.014799 Count:74
Insertion_Sort time:0.013301 > Merge_Sort time:0.012602 Count:75
Insertion_Sort time:0.013627 > Merge_Sort time:0.012961 Count:76
Insertion_Sort time:0.013998 > Merge_Sort time:0.013145 Count:77
Insertion_Sort time:0.014302 > Merge_Sort time:0.013177 Count:78
Insertion_Sort time:0.014683 > Merge_Sort time:0.013251 Count:79
Insertion_Sort time:0.014989 > Merge_Sort time:0.013374 Count:80
Insertion_Sort time:0.015425 > Merge_Sort time:0.013826 Count:81

Q2:

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.



Quick_Sort.

{10, 5, 7, 9, 8, 3}.

{3, 5, 7, 9, 8, 10}.

[return i=5]

Quicksort (A, p, q).

if p<q

r ← partition (A, p, q).;

Quicksort (A, p, r-1);

Quicksort (A, r+1, q);

② . Quicksort. (A, 0, 4).

{3, 5, 7, 9, 8}.

i j

pivot. all of the > x = A[p].

A[p] ⇔ A[i] = 3.

{3, 5, 7, 9, 8}.

return. i = 0.

③ . Quicksort. (A, r+1, q).

(A, 1, 4).

{5, 7, 9, 8}

same as above

⇒ {5, 7, 9, 8}.

return. i = 1

④ . Quick Sort. (A, 2, 4).

~~7, 8~~, {7, 9, 8}.

⇒ {7, 9, 8}

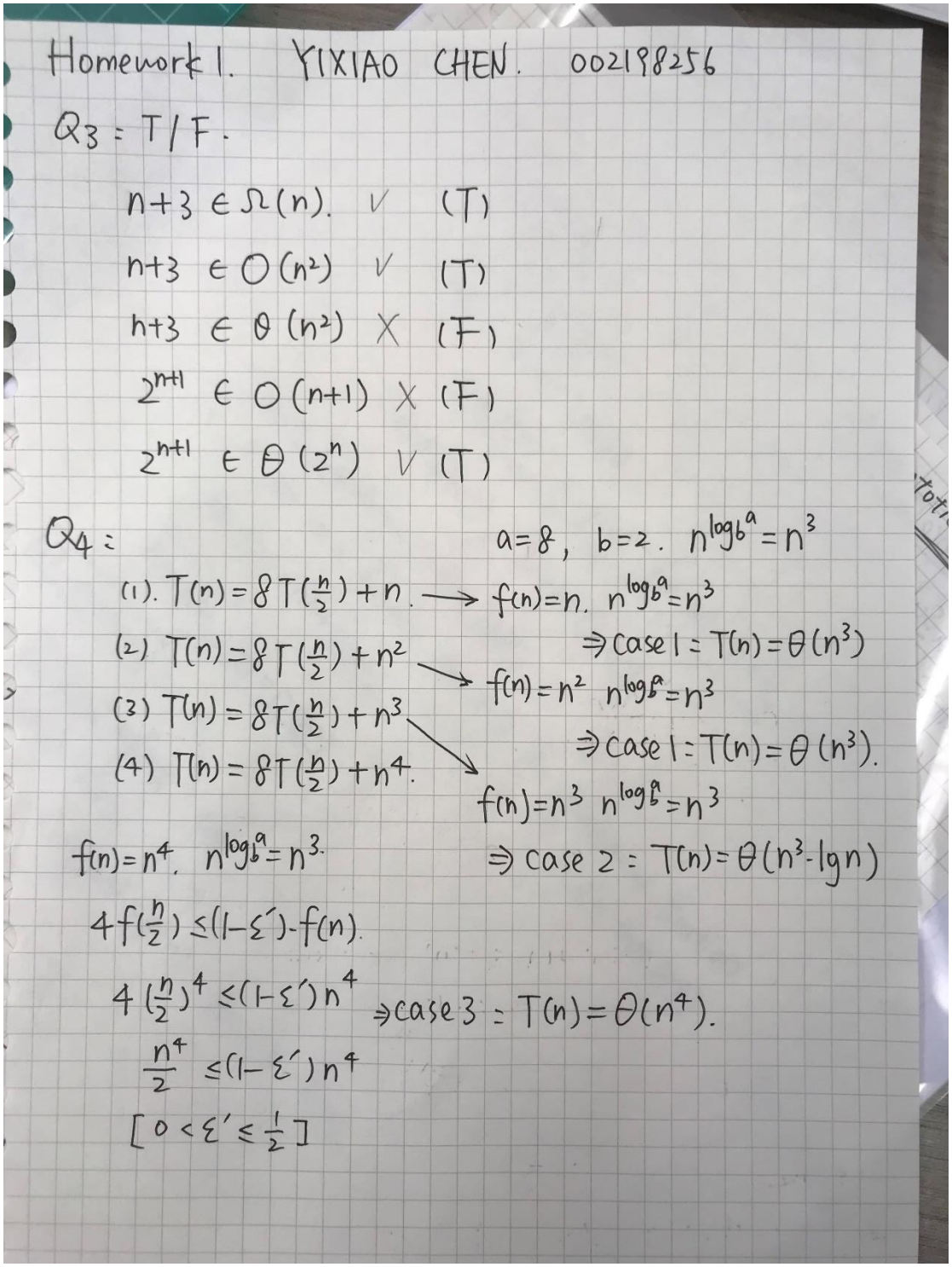
return. i = 2

⑤ . Quicksort. (A, 3, 4).

{9, 8} A[j] ≤ x = A[3] ⇒ i = i+1, A[i] ⇔ A[j]. ⇒ {~~9~~, 8}

Afinal = {3, 5, 7, 8, 9, 10}.

Q3/Q4：



Homework 1. YIXIAO CHEN. 002198256

Q3 : T / F.

$n+3 \in \Omega(n)$. ✓ (T)

$n+3 \in O(n^2)$ ✓ (T)

$n+3 \in \Theta(n^2)$ ✗ (F)

$2^{n+1} \in O(n+1)$ ✗ (F)

$2^{n+1} \in \Theta(2^n)$ ✓ (T)

Q4 :

$a=8,\ b=2.\ n^{\log_b a} = n^3$

(1). $T(n) = 8T(\frac{n}{2}) + n$. → $f(n)=n.\ n^{\log_b a}=n^3$

$\Rightarrow$ case 1 : $T(n)=\Theta(n^3)$

(2) $T(n) = 8T(\frac{n}{2}) + n^2$ → $f(n)=n^2\ n^{\log_b a}=n^3$

$\Rightarrow$ case 1 : $T(n)=\Theta(n^3)$.

(3) $T(n) = 8T(\frac{n}{2}) + n^3$ → $f(n)=n^3\ n^{\log_b a}=n^3$

$\Rightarrow$ case 2 : $T(n)=\Theta(n^3 \cdot \lg n)$

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

$f(n)=n^4.\ n^{\log_b a}=n^3.$

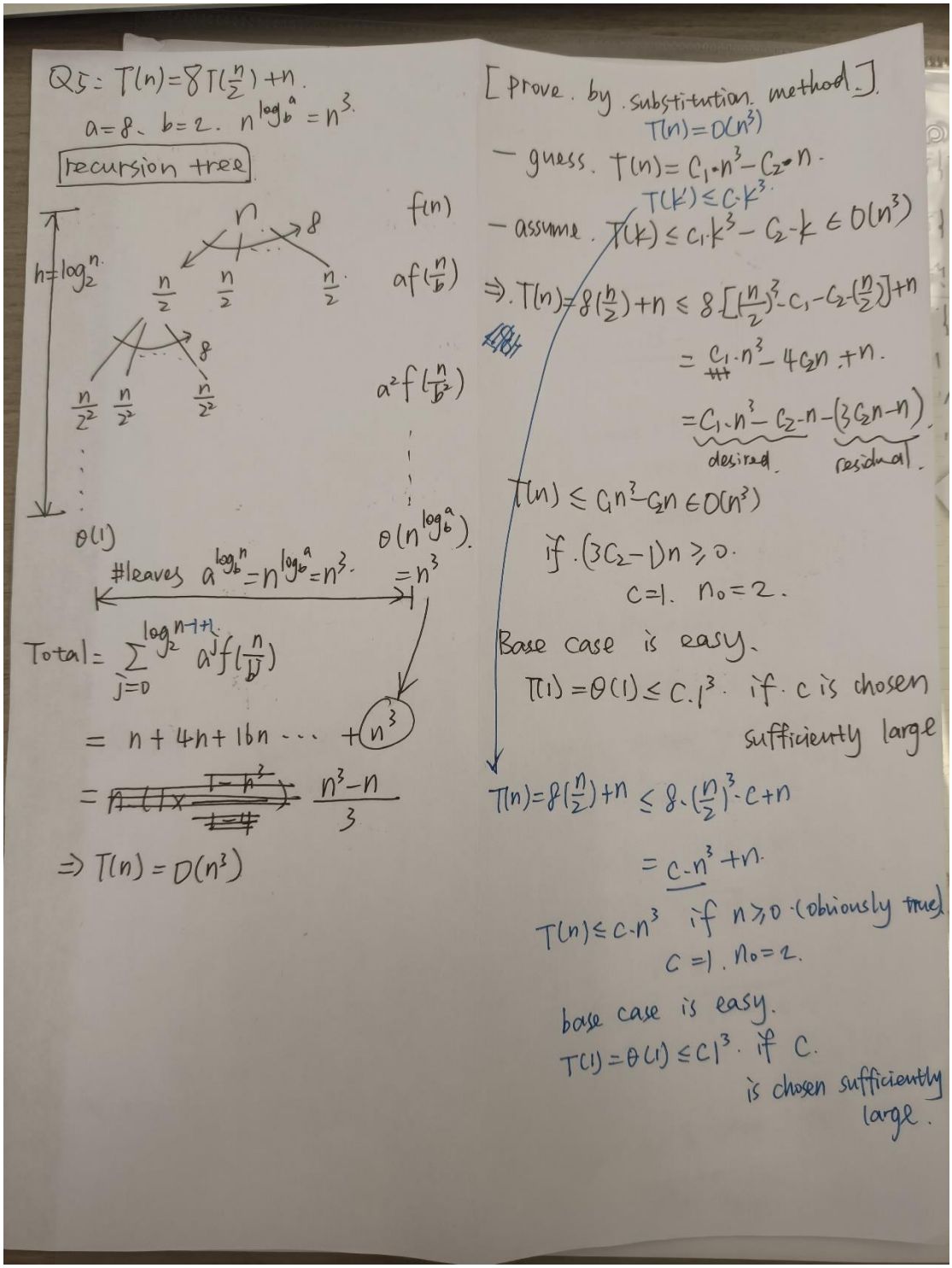
$4f(\frac{n}{2}) \le (1-\varepsilon')\cdot f(n)$.

$4(\frac{n}{2})^4 \le (1-\varepsilon')n^4$ $\Rightarrow$ case 3 : $T(n)=\Theta(n^4)$.

$\frac{n^4}{2} \le (1-\varepsilon')n^4$

$[0 < \varepsilon' \le \frac{1}{2}]$

Q5:


Q5: $T(n)=8T(\frac{n}{2})+n$.
$a=8,\ b=2,\ n^{\log_b a}=n^3$.
recursion tree
$h=\log_2 n$
$f(n)$
$af(\frac{n}{b})$
$a^2 f(\frac{n}{b^2})$
$\Theta(1)$
$\Theta(n^{\log_b a})$
$=n^3$
#leaves $a^{\log_b n}=n^{\log_b a}=n^3$.
Total $=\sum_{j=0}^{\log_2 n-1} a^j f(\frac{n}{b^j})$
$= n+4n+16n\cdots+n^3$
$=\frac{n^3-n}{3}$
$\Rightarrow T(n)=O(n^3)$
[prove by substitution method]
$T(n)=O(n^3)$
- guess. $T(n)=c_1 n^3-c_2 n$.
$T(k)\le c\cdot k^3$
- assume. $T(k)\le c_1 k^3-c_2 k \in O(n^3)$
$\Rightarrow T(n)=8(\frac{n}{2})+n \le 8[(\frac{n}{2})^3 c_1-c_2(\frac{n}{2})]+n$
$=c_1 n^3-4c_2 n+n$.
$=c_1 n^3-c_2 n-(3c_2 n-n)$
desired
residual
$T(n)\le cn^3-c_2 n \in O(n^3)$
if $(3c_2-1)n\ge 0$.
$c=1,\ n_0=2$.
Base case is easy.
$T(1)=\Theta(1)\le c\cdot 1^3$. if c is chosen sufficiently large
$T(n)=8(\frac{n}{2})+n \le 8\cdot(\frac{n}{2})^3\cdot c+n$
$=cn^3+n$.
$T(n)\le cn^3$ if $n\ge 0$ (obviously true)
$c=1,\ n_0=2$.
base case is easy.
$T(1)=\Theta(1)\le c1^3$. if c.
is chosen sufficiently large.