Algorithm Assignment 1

He Tianyang

March 13, 2024

1 Problem 1

Q15. Determine how many times the function Mult (see program 2-24) has executed multiplication, which implements the multiplication of two n×n matrices.

```
template <class T>
void Mult(T **a, T **b, T **c, int n)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            T sum = 0;
            for (int k = 0; k < n; k++)
                sum += a[i][k] * b[k][j];
            c[i][j] = sum;
        }
}
```

Program 2-24, Multiplication of two n×n matrices

Answer: Inside the K-loop, there are n multiplications. The k-loop is executed n times. J-loop is executed n times. I-loop is executed n times. So the total number of multiplications is $\mathbf{n^3}$.

2 Problem 2

Q16. Determine how many times the function Mult (see program 2-25) has executed multiplication, which implements the multiplication between an m×n matrix and an n×p matrix.

```
template <class T>
void Mult(T **a, T **b, T **c, int m, int n, int p)
{
    for (int i = 0; i < m; i++)
        for (int j = 0; j < p; j++)
        {
            T sum = 0;
            for (int k = 0; k < n; k++)
                sum += a[i][k] * b[k][j];
            c[i][j] = sum;
        }
}
```

Program 2-25, Multiplication between an m×n matrix and an n×p matrix

Answer: Inside the K-loop, there are n multiplications. The k-loop is executed n times. J-loop is executed p times. I-loop is executed m times. So the total number of multiplications is $\mathbf{m} \times \mathbf{n} \times \mathbf{p}$.

3 Problem 3

Q17. The function Min Max (see program 2-26) is used to find the maximum and minimum elements in the array a[0:n-1]. Let n be the instance characteristic. How many comparisons are there between the elements in a? Another function that finds the maximum and minimum elements is given in program 2-27. In the best and worst cases, how many comparisons are there between the elements in a? Analyze the relative performance between the two functions.

```cpp
template <class T>
bool MinMax(T a[], int n, T &min, T &max)
{
    if (n < 1) return false;
    min = max = a[0];
    for (int i = 1; i < n; i++)
    {
        if (a[i] < min) min = a[i];
        if (a[i] > max) max = a[i];
    }
    return true;
}
```

Program 2-26, Finding the maximum and minimum elements

```cpp
template <class T>
bool MinMax(T a[], int n, T &min, T &max)
{
    if (n < 1) return false;
    min = max = 0;
    for (int i = 1; i < n; i++)
    {
        if (a[min]>a[i]) min = i;
        else if (a[max]<a[i]) max = i;
    }
    return true;
}
```

Program 2-27, Alternative program to find the maximum and minimum elements

Answer:

1. In program 2-26, in single loop, there are 2 comparisons between the element in a. The loop is executed n-1 times. So the total number of comparisons is $\mathbf{2} \times (\mathbf{n} - \mathbf{1})$.
2. In program 2-27
   (a) For the best cases, the sequence is sorted in desand order. The first element is the maximum and the last element is the minimum. each element is compared once. So the total number of comparisons is $\mathbf{n} - \mathbf{1}$.
   (b) For the worst cases, the sequence is sorted in ascending order. The last element is the maximum and the first element is the minimum. each element is compared twice. So the total number of comparisons is $\mathbf{2} \times (\mathbf{n} - \mathbf{1})$.
3. Relative performance: the performance of the two functions is the same in the best cases. In the worst cases, the performance of program 2-27 is better as the performance of program 2-26. Therefore, **the performance of program 2-27 is better than the performance of program 2-26**.

4 Problem 4

Q20. Program 2-28 gives another iterative sequential search function. In the worst case, how many comparisons are there between x and the elements in a? Compare the number of comparisons with the corresponding number of comparisons in program 2-1, which function will run faster? Why?

```cpp
template <class T>
int SequentialSearch(T a[], const T &x, int n)
{
    // Search for x in the unsorted array a[0:n-1]
    // If found, return the corresponding position, otherwise return -1
    int i;
    for (i = 0; i < n && a[i] != x; i++);
    if (i == n) return -1;
    return i;
}
```

Program 2-1, Iterative sequential search function

```cpp
template <class T>
int SequentialSearch(T a[], const T &x, int n)
{
    // Search for x in the unsorted array a[0:n-1]
    // If found, return the corresponding position, otherwise return -1
    a[n] = x; // Assume this position is valid
    int i;
    for (i = 0; a[i] != x; i++);
    if (i == n) return -1;
    return i;
}
```

Program 2-28, Another iterative sequential search function

Answer:

- In the worst case, the number of comparisons between x and the elements in a is $\mathbf{n+1}$ in program 2-28.
- The number of comparisons between x and the elements in a is $\mathbf{n}$ in program 2-1.
- The function in program 2-28 will run faster. Because there is also the comparison between i and n, the total comparison in program 2-28 is $\mathbf{n+2}$ times when the worst case, and the total comparison in program 2-1 is $\mathbf{2}\times(\mathbf{n+1})$ times when the worst case. Therefore, **the function in program 2-28 will run faster**.

5 Problem 5

Q24. Calculate the average number of steps for the following functions:

1. SequentialSearch (see program 2-2)

```cpp
template <class T>
int SequentialSearch(T a[], const T &x, int n)
{
    // Search for x in the unsorted array a[0:n-1]
    // If found, return the corresponding position, otherwise return -1
    int i;
    for (i = 0; i < n && a[i] != x; i++);
    if (i == n) return -1;
    return i;
}
```

Program 2-2, Iterative sequential search function

2. SequentialSearch (see program 2-28)
3. Insert (see program 2-10)

```
template <class T>
void Insert(T a[], int &n, const T &x)
{
    // Insert element x into the ordered array a[0:n-1]
    // Assume that the size of a exceeds n
    int i;
    for (i = n - 1; i >= 0 && x < a[i]; i--)
        a[i + 1] = a[i];
    a[i + 1] = x;
    n++; // Added an element
}
```

Program 2-10, Insert element x into the ordered array

Answer: Donated one time of comparison of elements in a[0:n-1] as one step. The average number of steps is the average number of comparisons.

1. **SequentialSearch (Program 2-2)**: When target element **x** appear in position **i**, the number of steps is $\mathbf{i+1}$. When the target element not exist, the total steps is $\mathbf{n+1}$. Assume all the possible situation get the equal probability, therefore the average number of steps is $\frac{1}{n+1}\cdot(\sum_{i=0}^{n-1}(i+1)+n+1) = \boxed{\frac{n}{2}+1}$.
2. **SequentialSearch (Program 2-28)**: Target element **x** that app is position i needs $i+1$ steps, and the target element not exist needs $n+1$ steps. Assume all the possible situation get the equal probability, therefore the average number of steps is $\frac{1}{n+1}\cdot(\sum_{i=0}^{n-1}(i+1)+n+1) = \boxed{\frac{n}{2}+1}$.
3. **Insert (Program 2-10)**: There are n+1 possible insertion position. When insert into position before element i, the number of steps is $n-i+1$. When insert into the last position, the number of steps is 1. Assume all the possible situation get the equal probability, therefore the average number of steps is $\frac{1}{n+1}\cdot(\sum_{i=0}^{n-1}(n-i+1)+1) = \boxed{\frac{n+3}{2}}$.

6 Problem 6

Q37. Calculate the asymptotic complexity of the following functions and design a frequency table similar to Figures 2-19 to 2-22.

1. MinMax (see program 2-26)
2. Mult (see program 2-24)
3. Max (see program 1-31)
4. PolyEval (see program 2-3)
5. Horner (see program 2-4)
6. Rank (see program 2-5)
7. InsertionSort (see program 2-14)

6.1 Answers:

6.1.1 MinMax (Program 2-26)

```
template <class T>
bool MinMax(T a[], int n, T &min, T &max)
{
    if (n < 1) return false;
    min = max = a[0];
    for (int i = 1; i < n; i++)
    {
        if (a[i] < min) min = a[i];
        if (a[i] > max) max = a[i];
    }
    return true;
}
```

Program 2-26, Finding the maximum and minimum elements

Expressions	s/e	Frequency	Total
`template <class T>`	0	0	$\Theta(0)$
`void MinMax(T a[], int n, T &min, T &max) {`	0	0	$\Theta(0)$
`  if (n < 1) return false;`	1	1	$\Theta(1)$
`  min = max = a[0];`	1	1	$\Theta(1)$
`  for (int i = 1; i < n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`      if (a[i] < min) min = a[i];`	1	$\Omega(1)O(n)$	$\Omega(1)O(n)$
`      if (a[i] > max) max = a[i];`	1	$\Omega(1)O(n)$	$\Omega(1)O(n)$
`  }`	0	0	$\Theta(0)$
`  return true;`	1	1	$\Theta(1)$
`}`	0	0	0
total	$\Theta(n)$		

Table 1: Frequency table of MinMax

Therefore,

$$\boxed{t_{\text{MinMax}}(n) = O(n) = \Theta(n) = \Omega(n)} \tag{1}$$

6.1.2 Mult (Program 2-24)

```
template <class T>
void Mult(T **a, T **b, T **c, int n)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            T sum = 0;
            for (int k = 0; k < n; k++)
                sum += a[i][k] * b[k][j];
            c[i][j] = sum;
        }
}
```

Program 2-24, Multiplication of two n×n matrices

Expressions	s/e	Frequency	Steps
`template <class T>`	0	0	$\Theta(0)$
`void Mult(T **a, T **b, T **c, int n) {`	0	0	$\Theta(0)$
`  for (int i = 0; i < n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`    for (int j = 0; j < n; j++) {`	1	$\Theta(n^2)$	$\Theta(n^2)$
`      T sum = 0;`	1	$\Theta(n^2)$	$\Theta(n^2)$
`        for (int k = 0; k < n; k++) {`	1	$\Theta(n^3)$	$\Theta(n^3)$
`          sum += a[i][k] * b[k][j];`	1	$\Theta(n^3)$	$\Theta(n^3)$
`        }`	0	0	$\Theta(0)$
`        c[i][j] = sum;`	1	$\Theta(n^2)$	$\Theta(n^2)$
`    }`	0	0	$\Theta(0)$
`  }`	0	0	$\Theta(0)$
`}`	0	0	0
total	$\Theta(n^3)$		

Table 2: Frequency table of Mult

Therefore,

$$\boxed{t_{\text{Mult}}(n) = O(n^3) = \Theta(n^3) = \Omega(n^3)} \tag{2}$$

6.1.3 Max (Program 1-31)

```
template <class T>
int Max(T a[], int n)
{
    // Find the maximum element in a[0:n-1]
    int pos = 0;
    for (int i = 1; i < n; i++)
        if (a[pos] < a[i])
            pos = i;
    return pos;
}
```

Program 1-31, Finding the maximum element

Expressions	s/e	Frequency	Steps
`template <class T>`	0	0	$\Theta(0)$
`int Max(T a[], int n) {`	0	0	$\Theta(0)$
`  int pos = 0;`	1	1	$\Theta(1)$
`  for (int i = 1; i < n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`    if (a[pos] < a[i])`	1	$\Omega(1)O(n)$	$\Omega(1)O(n)$
`      pos = i;`	1	$\Omega(1)O(n)$	$\Omega(1)O(n)$
`  }`	0	0	$\Theta(0)$
`  return pos;`	1	1	$\Theta(1)$
`}`	0	0	0
total	$\Theta(n)$		

Table 3: Frequency table of Max

Therefore,

$$\boxed{t_{\text{Max}}(n) = O(n) = \Theta(n) = \Omega(n)} \tag{3}$$

6.1.4 PolyEval (Program 2-3)

```
template <class T>
T PolyEval(T coeff[], int n, const T &x)
{ // Calculate the value of an n-degree polynomial, coeff[0:n] is the coefficient of the
polynomial
    T y = 1, value = coeff[0];
    for (int i = 1; i <= n; i++)
    { // Accumulate the next term
        y *= x;
        value += y * coeff[i];
    }
    return value;
}
```

Program 2-3, Calculate the value of an n-degree polynomial

Expressions	s/e	Frequency	Steps
`template <class T>`	0	0	$\Theta(0)$
`T PolyEval(T coeff[], int n, const T &x) {`	0	0	$\Theta(0)$
`T y = 1, value = coeff[0];`	1	1	$\Theta(1)$
`for (int i = 1; i <= n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`y *= x;`	1	$\Theta(n)$	$\Theta(n)$
`value += y * coeff[i];`	1	$\Theta(n)$	$\Theta(n)$
`}`	0	0	$\Theta(0)$
`return value;`	1	1	$\Theta(1)$
`}`	0	0	0
total	$\Theta(n)$		

Table 4: Frequency table of PolyEval

Therefore,

$$\boxed{t_{\text{PolyEval}}(n) = O(n) = \Theta(n) = \Omega(n)} \tag{4}$$

6.1.5 Horner (Program 2-4)

```
template <class T>
T Horner(T coeff[], int n, const T &x)
{ // Calculate the value of an n-degree polynomial, coeff[0:n] is the coefficient of the
polynomial
    T value = coeff[n];
    for (int i = 1; i <= n; i++)
        value = value * x + coeff[n - i];
    return value;
}
```

Program 2-4, Calculate the value of an n-degree polynomial

Expressions	s/e	Frequency	Steps
`template <class T>`	0	0	$\Theta(0)$
`T Horner(T coeff[], int n, const T &x) {`	0	0	$\Theta(0)$
`  T value = coeff[n];`	1	1	$\Theta(1)$
`  for (int i = 1; i <= n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`    value = value * x + coeff[n - i];`	1	$\Theta(n)$	$\Theta(n)$
`  }`	0	0	$\Theta(0)$
`  return value;`	1	1	$\Theta(1)$
`}`	0	0	0
total	$\Theta(n)$		

Table 5: Frequency table of Horner

Therefore,

$$\boxed{t_{\text{Horner}}(n) = O(n) = \Theta(n) = \Omega(n)} \tag{5}$$

6.1.6 Rank (Program 2-5)

```
template <class T>
void Rank(T a[], int n, int r[])
{ // Calculate the rank of n elements in a[0:n-1]
    for (int i = 1; i < n; i++)
        r[i] = 0;
    // Initialize
    // Compare all elements in pairs
    for (int i = 1; i < n; i++)
        for (int j = 1; j < i; j++)
            if (a[j] <= a[i]) r[i]++;
            else r[j]++;
}
```

Program 2-5, Calculate the rank of n elements

Expressions	s/e	Frequency	Steps
`template <class T>`	0	0	$\Theta(0)$
`void Rank(T a[], int n, int r[]) {`	0	0	$\Theta(0)$
`  for (int i = 1; i < n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`    r[i] = 0;`	1	$\Theta(n)$	$\Theta(n)$
`  }`	0	0	$\Theta(0)$
`  for (int i = 1; i < n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`    for (int j = 1; j < i; j++) {`	1	$\Theta(n^2)$	$\Theta(n^2)$
`      if (a[j] <= a[i]) r[i]++;`	1	$\Omega(n)O(n^2)$	$\Omega(n)O(n^2)$
`      else r[j]++;`	1	$\Omega(n)O(n^2)$	$\Omega(n)O(n^2)$
`    }`	0	0	$\Theta(0)$
`  }`	0	0	$\Theta(0)$
`}`	0	0	0
total	$\Theta(n^2)$		

Table 6: Frequency table of Rank

Therefore,

$$\boxed{t_{\text{Rank}}(n) = O(n^2) = \Theta(n^2) = \Omega(n^2)} \tag{6}$$

6.1.7 InsertionSort (Program 2-14)

```
template <class T>
void Insert(T a[], int n, const T &x)
{ // Insert element x into the ordered array a[0:n-1]
    int i;
    for (i = n - 1; i >= 0 && x < a[i]; i--)
        a[i + 1] = a[i];
    a[i + 1] = x;
}
template <class T>
void InsertionSort(T a[], int n)
{ // Sort a[0:n-1]
    for (int i = 1; i < n; i++)
    {
        T t = a[i];
        Insert(a, i, t);
    }
}
```

Program 2-14, Insert element x into the ordered array

Expressions	s/e	Frequency	Steps
`template <class T>`	0	0	$\Theta(0)$
`void Insert(T a[], int n, const T &x) {`	0	0	$\Theta(0)$
`  int i;`	1	1	$\Theta(1)$
`  for (i = n - 1; i >= 0 && x < a[i]; i--)`	1	$\Omega(1)O(n)$	$\Omega(1)O(n)$
`    a[i + 1] = a[i];`	1	$\Omega(1)O(n)$	$\Omega(1)O(n)$
`  a[i + 1] = x;`	1	1	$\Theta(1)$
`}`	0	0	$\Theta(0)$
$t_{\text{Insert}}(n) = \Omega(1) = \Theta(n) = O(n)$			
`template <class T>`	0	0	$\Theta(0)$
`void InsertionSort(T a[], int n) {`	0	0	$\Theta(0)$
`  for (int i = 1; i < n; i++) {`	1	$\Theta(n)$	$\Theta(n)$
`    T t = a[i];`	1	$\Theta(n)$	$\Theta(n)$
`    Insert(a, i, t);`	$\Omega(1)O(n)$	$\Theta(n)$	$\Omega(n)O(n^2)$
`  }`	0	0	$\Theta(0)$
`}`	0	0	0
total	$\Theta(n^2)$		

Table 7: Frequency table of InsertionSort

Therefore,

$$\boxed{t_{\text{InsertionSort}}(n) = O(n^2) = \Theta(n^2) = \Omega(n)} \tag{7}$$