

## 2. Performance Analysis

### Problem 2.1

1. Show that  $5n^2 + 2n + 1$  is  $O(n^2)$
2. What is the Big Oh of  $n^2 + n \log(n)$ ? prove your answer.
3. Show that  $2n^3 \notin O(n^2)$ .
4. Assume that the expression below gives the processing time  $f(n)$  spent by an algorithm for solving a problem of size  $n$ .

$$10n + 0.1n^2$$

- (a) Select the dominant term(s) having the steepest increase in  $n$ .
  - (b) Specify the lowest Big-Oh complexity of the algorithm.
5. Determine whether each statement is *true* or *false* and correct the expression in the latter case:
    - (a)  $100n^3 + 8n^2 + 5n$  is  $O(n^4)$ .
    - (b)  $100n^3 + 8n^2 + 5n$  is  $O(n^2 \log n)$ .

### Problem 2.2

1. Find the simplest  $g(n)$ ,  $c$  and  $n_0$  for the following  $f(n)$  s.t:  $f(n) \leq cg(n), \forall n \geq n_0$ .
  - (a)  $6n^2 + n - 4$ .
  - (b)  $4 \log(n) + 2$ .
  - (c)  $3n^3 - 20n^2 + 10 \log(n)$ .
2. Find the big Oh notation for the following functions:
  - (a)  $n + \log(n^3) + n^2 \log(n)$ .
  - (b)  $2^{\log(n)+2} + 3^n$ .

### Problem 2.3

1. Order the following functions by asymptotic growth rate:  $4n \log n + 2n$ ,  $2^{10}$ ,  $2^{\log n}$ ,  $3n + 100 \log n$ ,  $4n$ ,  $2^n$ ,  $n^2 + 10n$ ,  $n^3$ ,  $n \log n$ . (Question R-4.8 page 182 of the textbook)

2. Show that  $\log n^{2n} + n^2$  is  $O(n^2)$
3. Show that  $\sum_{i=1}^5 i^3$  is  $O(1)$
4. Show that  $\sum_{i=1}^n \lceil \log i \rceil$  is a  $O(n \log n)$
5. Using the definition of the Big-Oh, prove that  $f(n) = 10n + 5 \log n$  is a big-oh of  $g(n) = n$ .

### Problem 2.4

Compute the following:

1.  $\sum_{i=0}^{n-1} 1.$
2.  $\sum_{i=i}^{n-1} i.$
3.  $\sum_{i=2}^{n-2} 1.$
4.  $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1.$
5.  $\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 1.$
6.  $\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1.$
7.  $\sum_{i=1}^{n-1} \sum_{j=i}^n 1.$
8.  $\sum_{i=1}^n \sum_{j=1}^n \sum_{k=0}^j 1.$

### Problem 2.5

Write the frequency for each line of the following code excerpts as a sum.

1. **for** (i = 1; i < n - 1; i++)

Sol.:  $(\sum_{i=1}^{n-2} 1) + 1$ . The +1 is for the last check.

2. **for** (i = n; i >= 0 ; i--)

3. **for** (i = 0; i < n ; i += 2)

4. **for** (i = 0; i < n ; i += 3)

5. **for** (i = 0; i < n ; i++)  
    **for** (j = 2; j < i; j++)

Sol. for line 2:  $(\sum_{i=0}^{n-1} \sum_{j=2}^{i-1} 1) + 1$ . The +1 is for the last check.

6. **for** (i = 0; i < n ; i++)  
    **for** (j = i; j > 0; j--)

7. **for** (i = 1; i <= n ; i \*= 2)

8. **for** (i = 1; i <= n ; i \*= 3)

### Problem 2.6

Analyze the following code excerpts:

1. **int** sum = 0;  
    **for** (**int** i = n; i > 0; i = i - 2)  
        sum = sum + i;

2. **int** sum = 0;  
    **for** (**int** i = 1; i < n; i = 2 \* i)  
        sum = sum + i;

3. **int** sum = 0;  
    **for** (**int** i = 1; i <= n ; i++)  
        **for** (**int** j = 0; j < 2 \* i ; j++)  
            sum += j;  
    **return** sum;

- 
4. 

```
for (int i = 0; i < n * n * n; i++) {
    System.out.println(i);
    for (int j = 2; j < n; j++)
        System.out.println(j); }
System.out.println("End!");
```
  5. 

```
int k = 100, sum = 0;
for (int i = 0; i < n; i++)
    for (j = 1; j <= k; j++) {
        sum = i + j;
        System.out.println(sum);
    }
```
  6. 

```
int sum = 0;
for(int i = 0; i < n * n; i++) {
    for(int j = n - 1; j >= n - 1 - i; j--) {
        sum = i + j;
        System.out.println(sum);
    }
}
```
  7. 

```
int sum = 0;
for(int i = 1; i <= 2^n; i = i * 2) {
    for(int j = 0; j <= log(i); j++) {
        sum = i + j;
        System.out.println(sum);
    }
}
```
  8. 

```
int sum = 0; int k = 2^3 ;
for(int i = k; i <= 2^(n - k); i = i * 2) {
    for(int j = 2^(i - k); j < 2^(i + k); j = j * 2) {
        sum = i + j;
        System.out.println(sum);
    }
}
```
  9. 

```
int sum = 0;
for(int i = 2^n; i >= 1; i = i / 2) {
    for(int j = i; j >= 1; j = j / 2) {
        sum = i + j;
        System.out.println(sum);
    }
}
```
  10. 

```
int sum = 0;
for(int i = n; i > 0; i--) {
    for(int j = i; j <= n; j++) {
        sum = i + j;
        System.out.println(sum);
    }
}
```
  11. 

```
int sum = 0;
for(int i = 0; i < n; i++) {
    for(int j = 0; j < i; j++) {
        for(int k = n; k > 0; k--)
            sum = i + j + k;
    }
}
```

- ```
12. int k = 1;
    for(int i = 1; k <= n; i *= ++k) {
        for(int j = 0; j < n; j++)
            sum = i + j;
    }
```
- ```
13. int sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++)
            sum = sum + A[j];
        A[i] = A[i] + sum;
    }
```
- ```
14. int k = 3, j = 5, sum = 0;
    for (int i = 0; i < n; i++)
        for (j = 1; j <= k; j++) {
            sum = i + j;
            System.out.println(sum);
        }
```
- ```
15. for (int i = 0; i < n * n * n; i++) {
    System.out.println(i);
    for (int j = 2; j < n; j++) {
        System.out.println(j);
    }
}
System.out.println("Goodbye!");
```
- ```
16. for (int i = 0; i < n * n; i++) {
    System.out.println(i);
    for (int j = 4; j <= n; j++) {
        System.out.println(j);
    }
}
System.out.println("Goodbye!");
```
- ```
17. m = 1;
    while( m < 100 ) {
        system.out.println(m);
        i = 0;
        while (i < n) {
            system.out.println( n * m);
            i++;
        }
        m++;
    }
```
- ```
18. for(int i = 0; i < 2 * n; i = i + 2) {
    for(int j = 0; j < n; j++)
        if (j % 2 == 0)
            system.out.println(j);
}
```
- ```
19. for (int i = 0; i < n * log(n); i++) {
    System.out.println(i);
    for (int j = 2; j < n; j++) {
        System.out.println(j);
    }
}
```

- ```

20. for (int i = 0; i < n * n; i++) {
    System.out.println(i);
    for (int j = 2 * n; j > n; j--) {
        System.out.println(j);
    }
}

21. int m = 1;
    while( m <= n ) {
        system.out.println(m);
        i = n;
        while (i > 0 ) {
            system.out.println(i);
            i = i / 2;
        }
        m++;
    }

22. for(int i = 0; i < 2 * n; i = i + 2) {
    for(int j = 0; j < i; j++)
        if (j % 2 == 0)
            system.out.println(j);
}

23. int i = 1;
    while (i < n) {
        i++;
        if (i > 7) break;
    }

24. int A = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 0; j < min(i, n / 2); j++)
            A++;

```

### Problem 2.7

1. Given an  $n$ -element array  $X$ , Algorithm  $B$  chooses  $\log n$  elements in  $X$  at random and executes an  $O(n)$ -time calculation for each. What is the worst-case running time of Algorithm  $B$ ? (Question R-4.30 page 184 of the textbook)
2. Given an  $n$ -element array  $X$  of integers, Algorithm  $C$  executes an  $O(n)$ -time computation for each even number in  $X$ , and an  $O(\log n)$ -time computation for each odd number in  $X$ . What are the best-case and worst-case running times of Algorithm  $C$ ? (Question R-4.31 page 184 of the textbook)

### Problem 2.8

Give in asymptotic notation the running time for the following algorithms:

1. Vector-vector addition (the vectors are of size  $n$ ).
2. Dot product of two vectors (the vectors are of size  $n$ ).
3. Matrix-vector multiplication (the matrix is of size  $m \times n$ , the vector is of size  $n$ ).
4. Matrix addition (the two matrices are of size  $m \times n$ ).
5. Matrix-Matrix multiplication (the two matrices are of size  $m \times k$  and  $k \times n$  respectively).

### Problem 2.9

For the following functions:

1. Give two example inputs leading to the best and worst running time respectively.
2. Analyze the performance of the function in each case (best and worst).

```
public int func1 (int A[], int n) {
    int maxr= 0;
    int maxi= 0;
    int i= 0;
    while (i < n) {
        int j= i+1;
        int nbr= 1;
        while ((j < n) && (A[i] == A[j])) {
            nbr++;
            j++;
        }
        if (nbr > maxr) {
            maxr= nbr;
            maxi= i;
        }
        i= j;
    }
    return maxi;
}
```

```
public int func2 (int A[], int n) {
    int maxr= 0;
    int maxi= 0;
    int i= 0;
    while (i < n) {
        int j= i+1;
        int nbr= 1;
        while (j < n) {
            if (A[i] == A[j])
                nbr++;
            j++;
        }
        if (nbr > maxr) {
            maxr= nbr;
            maxi= i;
        }
        i++;
    }
    return maxi;
}
```

```
public void func3 (int A[], int n) {
    int i= 0;
    int j= n-1;
    while (i < j) {
        while ((A[i] <= 0) && (i<j)) {
            i++;
        }
        while ((A[j] > 0) && (i<j)) {
            j--;
        }
        int tmp= A[i];
        A[i]= A[j];
        A[j]= tmp;
    }
}
```

```
public void func4(int A[], int B[], int C[], int n) {
```

```

int i= 0;
int j= 0;
int k= 0;
while ((i < n) && (j < n)){
    if (A[i] <= B[j])
        C[k++] = A[i++];
    else
        C[k++] = B[j++];
}
if (i == n) {
    while(j < n)
        C[k++] = B[j++];
}
else {
    while(i < n)
        C[k++] = A[i++];
}
}

```

### Problem 2.10

The space performance (or complexity) of an algorithm is the maximum amount of memory (in bytes) used at any point of the algorithm **ignoring the input size**.

■ **Example 2.1** The function `sum1` below uses two variables (`sum` and `i`) in addition to the input `A`, so it is  $O(1)$  in space (and  $O(n)$  in time).

```

int sum1(int[] A, int n) {
    int sum = 0;
    for(int i = 0; i < n; i++) {
        sum += A[i];
    }
    return sum;
}

```

On the other hand, the function `sum2` is  $O(n)$  in space (why?):

```

int sum2(int[] A, int n) {
    int sum = 0;
    for(int i = 0; i < n; i++) {
        int[] B = new int[i + 1];
        for(int j = i; j <= i; j++) {
            B[j] = A[j] - A[i];
        }
        for(int j = i; j <= i; j++) {
            sum += B[j];
        }
    }
    return sum;
}

```

■

What is the space complexity of the following function? Justify your answer.

```

public void func3 (int A[], int n) {
    int i= 0;
    int j= n-1;
    while (i < j) {
        while ((A[i] <= 0) && (i<j)) {
            i++;
        }
    }
}

```

```

        while ((A[j] > 0) && (i < j)) {
            j--;
        }
        int tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;
    }
}

```

### Problem 2.11

The class *Sort* below implements three sorting algorithms: selection sort, bubble sort and Quicksort.

```

import java.util.Arrays;

public class Sort {
    public static void selectionSort(double[] A, int n) {
        for (int i = 0; i < n - 1; i++) {
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (A[j] < A[min])
                    min = j;
            }
            double tmp = A[i];
            A[i] = A[min];
            A[min] = tmp;
        }
    }

    public static void bubbleSort(double A[], int n) {
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (A[j] < A[j + 1]) {
                    double tmp = A[j];
                    A[j] = A[j + 1];
                    A[j + 1] = tmp;
                }
            }
        }
    }

    public static void quickSort(double A[], int n) {
        Arrays.sort(A, 0, n - 1);
    }
}

```

Conduct an experimental analysis of these three algorithms as follows:

- Use arrays of sizes ranging from 10000 to 50000 with step size 10000 (so in total you have 5 different sizes).
  - Give the same input to all three algorithms.
  - Fill the array with random numbers (use `Math.random()`).
  - For each input repeat the execution 100 times, measure the execution in nanoseconds (use `System.nanoTime()`), and report the average time in milliseconds.
1. Write the code used for the experimental analysis.
  2. Report the results as a table and as a graph.
  3. Which of the three algorithms is the fastest?
  4. Which of *selection sort* and *bubble sort* is faster? Which one has a larger growth rate?



**Problem 2.12**

Use the definition to show that:

1.  $\log_a(n) \in O(\log_b(n))$ ,  $\forall a, b > 1$ . (Changing the base of the logarithm **does not** change the growth rate)
2.  $a^n \notin O(b^n)$ ,  $\forall a > b > 0$ . (Changing the base of the exponential **does** change the growth rate)

**Problem 2.13**

1. Find the best asymptotic notation for the following functions:
  - (a)  $\log(n^{n^2}) + n^2 \log(n^{\log n}) + n^2$ .
  - (b)  $2^n + 2^{\log(n!) + \log n}$ .
2. Show the following:
  - (a)  $\sum_{i=1}^n i^2$  is  $O(n^3)$ .
  - (b)  $\sum_{k=0}^{n-1} \log(n-k)$  is  $O(n \log n)$ .
  - (c)  $\sum_{i=0}^{\log n - 1} 2^i (\log n - i)$  is  $O(n)$ .

**Problem 2.14**

Use the definition to show that:

1.  $\forall c \in \mathbb{R}, cf \in O(f)$ .
2. If  $\exists n_0 \geq 0$ , such that  $f(n) \leq g(n), \forall n \geq n_0$ , then  $f + g \in O(g)$ .
3. If  $f \in O(g)$  and  $g \in O(h)$ , then  $f \in O(h)$ .

**Problem 2.15**

Show that:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \implies f \in O(g) \text{ and } g \notin O(f); \\ c > 0 & \implies f \in O(g) \text{ and } g \in O(f); \\ \infty & \implies f \notin O(g) \text{ and } g \in O(f). \end{cases}$$