CHITKARA

DATA STRUCTURES AND ALGORITHMS 2023

ONTRACK SUBMISSION

Task 2.1

Submitted By:
Rudraksh DHAMIJA
2210994829
2023/09/13 19:14

Tutor:
Angra SHEENA

Outcome	**Weight**
Evalation and Compuation of Memory	♦◊◊◊◊
Create Data Structure	♦◊◊◊◊
Creating Software Solutions	♦♦◊◊◊

DSA

September 13, 2023

Practical Task 2.1

Name: Rudraksh

Roll Number: 2210994829

1. The following table contains six example algorithms. Each algorithm provides an example of the operations per line. The lines in red show operation that always occur, while the blue ones only occur in the worst case. Each algorithm has also calculated the total operations in the Best (B) and average

(A) cases. Read each to understand how they are calculated

	Algorithm	Operation count per line	Total

i.	`num = rand();`
	`double a = num;`
	`if( a < 0.5 ) a += num;`
	`if( a < 1.0 ) a += num;`
	`if( a < 1.5 ) a += num;`
	`if( a < 2.0 ) a += num;`
	`if( a < 2.5 ) a += num;`
	`if( a < 3.0 ) a += num;`
		1
		1
		1 + 1
		1 + 1
		1 + 1
		1 + 1
		1 + 1
		1 + 1
			W = ?
			B = 8
			A = 11

ii.	`int count = 0;`
	`for (int i = 0; i < N; i++){`
	`    if( rand() < 0.5 ){`
	`        count += 1;`
	`    }`
	`}`
		1
		1 + (N+1) + N
		$N \times (1)$
		$N \times (1)$
			W = ?
			B = 3N+3
			A = $\frac{7N}{2}+3$

iii.	`int count = 0;`
	`for (int i = 0; i < N; i++) {`
	`    if (unlucky){`
	`        for (j = N; j > i; j--){`
	`            count = count + i + j;`
	`        }`
	`    }`
	`}`
		1
		1 + (N+1) + N
		$N \times (1)$
		$N \times (1 + \left(\frac{N+1}{2} + 1\right) + \frac{N+1}{2})$
		$N \times \frac{N+1}{2} \times (1)$
			W =
			B = $3N+3$
			A = $\frac{3N^2}{4}+\frac{19N}{4}+3$

iv.	`int count = 0;`
	`int i = N;`
	
	`if (unlucky) {`
	`    while (i > 0){`
	`        count += i;`
	`        i /= 2;`
	`    }`
	`}`
		1
		1
		
		1
		Log N + 1
		Log N
		Log N
			W = ?
			B = 3
			A = $\frac{3logN}{2}+4\frac{1}{2}$

v.	`int count = 0;`
	
	`for (int i = 0; i < N; i++) {`
	`    int num = rand();`
	`    if( num < 0.5 ) {`
	`        count += 1;`
	`    }`
	`}`
	`int num = count;`
	`for (int j = 0; j < num; j++) {`
	`    count = count + j;`
	`}`
		1
		
		1 + (N+1) + N
		$N \times (1)$
		$N \times (1)$
		$N \times (1)$
		
		1
		1 + (num + 1) + num
		$num \times (1)$
			W = ?
			B = $4N+6$
			A = $6N+6$

vi.	`for (int i = 0; i < N - 1; i++){`
	`    for (int j = 0; j < N-i-1; j++){`
	`        if (a[j] > a[j+1]){`
	`            Swap(a[j], a[j + 1]);`
	`        }`
	`    }`
	`}`
		1 + N + (N-1) = 2N
		$(N-1) \times (1 + (\frac{N}{2}+1) + \frac{N}{2})$ = $N^2 - N - 2$
		$(N-1) \times \left(\frac{N}{2}\right) \times (1)$ = $\frac{N^2}{2} - \frac{N}{2}$
		$(N-1) \times \left(\frac{N}{2}\right) \times (1)$ = $\frac{N^2}{2} - \frac{N}{2}$
			W = ?
			B = $\frac{3N^2}{2}+\frac{N}{2}-2$
			A = $\frac{7N^2}{4}+\frac{N}{4}-2$

A. **First go through the above table and calculate the worst case for each algorithm.** i. 14

ii. 4N+3

iii. (3N^2+13N+6)/2

iv. 3logn+4

v. 8N+6

vi. 2N^2-2

B. **Describe the equations found in (a) for the best, worst and average cases using Big-Θ notation.**

i. Best- Θ (1)
Worst- Θ (1)
Average- Θ (1)

ii. Best- Θ (n)
Worst- Θ (n)
Average- Θ (n)

iii. Best- Θ (n)
Worst- Θ (n^2)
Average- Θ (n^2)

iv. Best- Θ (1)
Worst- Θ (logn)
Average- Θ (logn)

v. Best- Θ (n)
Worst- Θ (n)
Average- Θ (n)

vi. Best- Θ (n^2)
Worst- Θ (n^2)
Average- Θ (n^2)

C. **Describe each algorithm's overall performance using the tightest possible class in Big-O notation**

i. O(1)

ii. O(n)

iii. O(n^2)

iv. O(logn)

v. O(n)

vi. O(n^2)

D. **Describe each algorithm's overall performance using the tightest possible classin Big- Ω notation**

i. Ω (1)

ii. Ω (n)

iii. Ω (n)

iv. Ω (1)

v. Ω (n)

vi. Ω (n^2)

E. Describe each algorithm's overall performance using Big-Θ notation

i. Θ (1)
ii. Θ (n)
iii. can't determine
iv. can't determine
v. Θ (n)
vi. Θ (n^2)

F. Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation

By focusing on each algorithm's worst-case time complexity using Big-0 notation, it is easiest to quickly summarize each algorithm's effectiveness using asymptotic notation. Combining the algorithm's worst-case time complexity allows you to quickly and effectively demonstrate how the method becomes more efficient as the input size increases.

i) Θ (1)

ii) Θ (n)

iii) Θ (n^2)

iv) Θ (logn)

v) Θ (n)

vi) Θ (n^2)

2. Arguably, the most commonly used asymptotic notation used is Big-O. Discuss why this is so commonly the case and why more people do not use Big- Θ

Big-O notation makes it simple to calculate the upper bound for any algorithm, which is extremely helpful because it assures us that the algorithm won't perform worse than that. The Big-theta notation, on the other hand, does not offer worst-case analysis and may not give us any useful information when Big-O and Big-omega are not equal.

3. Is it true that Θ(n2) algorithm always takes longer to run than an Θ (log n) algorithm? Explain your answer

It is true that algorithms with time complexity of (n2) will typically execute more slowly than those with time complexity of (log n). The number of operations increases as the size of the input does because (n2) shows a quadratic growth rate. That proves that as the number of inputs rises, more operations are needed, which necessitates a longer execution time. In contrast, the growth rate of (n log n), where the number of operations grows at a much slower rate, is logarithmic. As n grows, more operations are carried out, which speeds up completion. (Log n) is effective for large inputs, while (n2) is effective for small inputs.

4. Answer whether the following statements are right or wrong and explain your answers

i. n^2 + (6^13)n = 0(n^2)

TRUE: The influence of the quadratic component n ^2 will eventually surpass that of the term (613)n as n increases in value. Therefore, Big O(n2) is a suitable notation to represent this behavior.

ii. nlogn = 0(n)
False: nlogn can't be Big O(n), it will always be Big O(nlogn)

iii. n^2 + n + 10^6 = Θ(n^2)
False: Since n ^2 is the main term in the equation, both the Big O and Big Ω notations apply to it. Since n2 is dominant and n ^3 cannot be the Big notation, the proper Big Θ notation is n2.

iv. nlogn + 51n^2 = Ω(n)
TRUE: nlogn will serve as the equation's Big Ω notation. But it's crucial to remember that Big indicates the loosest bottom bound, leaving room for options like Big (n), Big (logn), or Big Ω (1).