Practical Task 2.1

(Pass Task)

Submission deadline: 11:59 pm Sunday, March 28st
Discussion deadline: 11:59 pm Sunday, April 18th

Task Objective

In this task you will use knowledge and skills of algorithm space and time complexity to answer a series of questions.

Background

The way we write code affects how fast that code runs and the amount of memory it uses. When only dealing with small amounts of data the efficiency does not matter very much. However, as the amount of data increases, such as those in most real-world applications, so does the importance of our choice of algorithm. The aim of this unit is to ensure we write solutions to problems that run as efficiently as possible. Therefore, we need to be able to measure how efficient each algorithm is in-order to compare our options and select the right solution for the problem.

We define complexity as the time it will take to solve for an input size of n elements, often expressed with the function $T(n)$. We want to be able to define this time such that it does not depend on the implementation used. This means when finding the complexity of an algorithm in general we want to remove individual implementation factors, such as processor speed, instruction set, disk speed, compiler, bus speed etc and focus just on the algorithm itself. We can do this evaluation using one of several methods: Operation Counting, Asymptotic Notations, Substitution Method, Recurrence Tree, Master Method. In this unit we will look at the first two approaches, while SIT320 Advanced Algorithms will look at the last the three methods.

Operation Counting is a process where we calculate the amount of times each operation in an algorithm is run giving us the total number of operations. This results in a complex function that may not always be easy to use when comparing algorithms. Therefore, we typically estimate complexity of each algorithm *asymptotically.* Asymptotic notation aims to *classify* algorithms according to their performance. This means, rather than find the exact efficiency (as in Operation Counting) we want to group algorithms that behave similarly. This categorisation is done based on the number of elementary *steps*, taken in constant time, the algorithm takes to complete.

There are different approaches to classifying the time function depending on the way the behaves as it approaches a limit point, eg positive or negative infinity or zero. These are based around the Upper Bound (called Big-O), the Lower Bound (called Big Omega) and the Tightly Bound (called Big Theta) notations. Frequently, documents will often only refer to the Upper Bound (Big-O) complexity of an algorithm. Using this notation there are a number of classifications that an algorithm is typically placed, such as (in increasing complexity) $O(1)$, $O(\log_2 n)$, $O(n)$, $O(\mathrm{n} \log_2 n)$, $O(n^2)$, $O(n^2 \ \log_2 n)$, $O(n^3)$, $O(2^n)$ and $O(n!)$.

When analysing an algorithm, we may wish to determine the complexity of the algorithm in different situations representing different operational situations. For instance, it is common to consider the best, worst and average situations. For example, we may want to know how well a search algorithm will perform if the first item it looks at is the item it is searching for. We may also want to understand what happens if it is the last item in the array and frequently we want to know how it will perform on average. With many algorithms the average complexity is the same as the worst case.

You should review the lecture and read up on Asymptotic Notations and complexity then answer each of the questions on this sheet.

Questions

In this task, answer all the following questions and complement each answer with an explanation.

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();`	1	W = ?
	`double a = num;`	1	B = 8
	`if( a < 0.5 ) a += num;`	1 + 1	A = 11
	`if( a < 1.0 ) a += num;`	1 + 1	
	`if( a < 1.5 ) a += num;`	1 + 1	
	`if( a < 2.0 ) a += num;`	1 + 1	
	`if( a < 2.5 ) a += num;`	1 + 1	
	`if( a < 3.0 ) a += num;`	1 + 1	
ii.	`int count = 0;`	1	W = ?
	`for (int i = 0; i < N; i++){`	1 + (N+1) + N	B = 3N+3
	`    if( rand() < 0.5 ){`	$N \times (1)$	A = $\frac{7N}{2} + 3$
	`        count += 1;`	$N \times (1)$	
	`    }`		
	`}`		
iii.	`int count = 0;`	1	W =
	`for (int i = 0; i < N; i++) {`	1 + (N+1) + N	B = $3N + 3$
	`    if (unlucky){`	$N \times (1)$	A = $\frac{3N^2}{4} + \frac{19N}{4} + 3$
	`        for (j = N; j > i; j--){`	$N \times (1 + \left(\frac{N+1}{2} + 1\right) + \frac{N+1}{2})$	
	`            count = count + i + j;`	$N \times \frac{N+1}{2} \times (1)$	
	`        }`		
	`    }`		
	`}`		
iv.	`int count = 0;`	1	W = ?
	`int i = N;`	1	B = 3
			A = $\frac{3logN}{2} + 4\frac{1}{2}$
	`if (unlucky) {`	1	
	`    while (i > 0){`	Log N + 1	
	`        count += i;`	Log N	
	`        i /= 2;`	Log N	
	`    }`		
	`}`		
v.	`int count = 0;`	1	W = ?
			B = $4N + 6$
	`for (int i = 0; i < N; i++) {`	1 + (N+1) + N	A = $6N + 6$
	`    int num = rand();`	$N \times (1)$	
	`    if( num < 0.5 ) {`	$N \times (1)$	
	`        count += 1;`	$N \times (1)$	
	`    }`		
	`}`		
	`int num = count;`	1	
	`for (int j = 0; j < num; j++) {`	1 + (num + 1) + num	
	`    count = count + j;`	$num \times (1)$	
	`}`		
vi.	`for (int i = 0; i < N - 1; i++){`	1 + N + (N-1) = 2N	W = ?
	`    for (int j = 0; j < N-i-1; j++){`	$(N-1) \times (1 + \frac{N}{2} + \frac{N-1}{2}) = N^2 - \frac{N+1}{2}$	B = $\frac{3N^2}{2} + \frac{N}{2}$
	`        if (a[j] > a[j+1]){`	$(N-1) \times \left(\frac{N-1}{2}\right) \times (1) = \frac{N^2}{2} - N + \frac{1}{2}$	A = $\frac{7N^2}{4} + \frac{1}{4}$
	`            Swap(a[j], a[j + 1]);`	$(N-1) \times \left(\frac{N-1}{2}\right) \times (1) = \frac{N^2}{2} - N + \frac{1}{2}$	
	`        }`		
	`    }`		
	`}`		

For each of the above algorithms answer the following sub-questions for both algorithms.

a) First go through the above table and calculate the worst case for each algorithm.
b) Describe the equations found in (a) for the **best**, **worst** and **average** cases using Big-Θ notation.
c) Describe each algorithm's overall performance using the tightest possible class in Big-O notation.
d) Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.
e) Describe each algorithm's overall performance using Big-Θ notation.
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.

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.
3. Is it true that $\Theta(n^2)$ algorithm always takes longer to run than an $\Theta(\log n)$ algorithm? Explain your answer.
4. Answer whether the following statements are right or wrong and explain your answers.

   $n^2 + 6^{13}n = O(n^2)$
   $nlog\, n = O(n)$
   $n^2 + n + 10^6 = \Theta(n^3)$
   $n\, log\, n + 51n^2 = \Omega(n)$

Further Notes

- You may find the attached *Asymptotic Cheat Sheet* useful to clarify the mathematical notation.
- You will find ultimate answers to these tasks by exploring chapters 4.1.-4.3 of the course book "Data Structures and Algorithms in Java" by Michael T. Goodrich, Irvine Roberto Tamassia, and Michael H. Goldwasser (2014). You may access the book on-line for free from the reading list application in CloudDeakin available in Resources → Course materials → Course Book: Data structures and algorithms in Java.

Marking Process and Discussion

To get your task completed, you must finish the following steps strictly on time.

1. Work on your task either during your allocated lab time or during your own study time.
2. Once the task is complete you should make sure that your program implements all the required functionality, is compliable, and has no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your program thoroughly before submission. Think about potential errors where your program might fail. Note we can sometime use test cases that are different to those provided so verify you have checked it more thoroughly than just using the test program provided.
3. Submit your solution as an answer to the task via the OnTrack submission system. This first submission must be prior to the submission "S" deadline indicated in the unit guide and in OnTrack.
4. If your task has been assessed as requiring a "Redo" or "Resubmit" then you should prepare a new submission. You will have 1 (7 day) calendar week from the day you receive the assessment from the tutor. This usually will mean you should revise the lecture, the readings indicated, and read the unit discussion list for suggestions. After your submission has been corrected and providing it is still before the due deadline you can resubmit.
5. If your task has been assessed as correct, either after step 3 or 4, you can "discuss" with your tutor. This first discussion must occur prior to the discussion "D".

6. Meet with your tutor or answer question via the intelligent discussion facility to demonstrate/discuss your submission. Be on time with respect to the specified discussion deadline.
7. The tutor will ask you both theoretical and practical questions. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this compulsory interview part. The tutor will tick off the task as complete, only if you provide a satisfactory answer to these questions.
8. If you cannot answer the questions satisfactorily your task will remain on discussion and you will need to study the topic during the week and have a second discussion the following week.
9. Please note, due to the number of students and time constraints tutors will only be expected to mark and/or discuss your task twice. After this it will be marked as a “Exceeded Feedback”.
10. If your task has been marked as “Exceeded Feedback” you are eligible to do the redemption quiz for this task. Go to this unit’s site on Deakin Sync and find the redemption quiz associated with this task. You get three tries at this quiz. Ensure you record your attempt.
    I. Login to Zoom and join a meeting by yourself.
    II. Ensure you have both a camera and microphone working
    III. Start a recording.
    IV. Select Share screen then select “Screen”. This will share your whole desktop. Ensure Zoom is including your camera view of you in the corner.
    V. Bring your browser up and do the quiz.
    VI. Once finished select stop recording.
    VII. After five to ten minutes you should get an email from Zoom providing you with a link to your video. Using the share link, copy this and paste in your chat for this task in OnTrack for your tutor to verify the recording.
11. Note that we will not check your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work through the unit.
12. Final note, A “Fail” or “Exceeded Feedback” grade on a task does not mean you have failed the unit. It simply means that you have not demonstrated your understanding of that task through OnTrack. Similarly failing the redemption quiz also does not mean you have failed the unit. You can replace a task with a task from a higher grade.