W03 - Efficiency and O-notation/ Array Implementation/ Amortized Analysis

Due 9 Feb at 23:59 **Points** 20 **Questions** 13
Available until 9 Feb at 23:59 **Time limit** None **Allowed attempts** Unlimited

Instructions

Content and Background

This quiz relates to the content covered in the course up till now. It may also draw upon supporting knowledge and skills expected from a CS sophomore. Please make sure that you are up to date on the course work before attempting the quiz.

Difficulty

This quiz is equivalent to an in-class exercise. Have pen and paper ready and be prepared to work on challenging problems.

Take the quiz again

Attempt history

	Attempt	Time	Score
LATEST	Attempt 1	10 minutes	20 out of 20

Correct answers are hidden.

Score for this attempt: **20** out of 20
Submitted 9 Feb at 13:37
This attempt took 10 minutes.

Question 1 **2 / 2 pts**

Mark the statements below that are true.

- [ ] $(4n + 6)$ is in $O(4)$
- [x] $(4n^3)$ is in $O(4n^3)$
- [x] $(\log_{10} n)$ is in $O(\log_2 n)$
- [x] $(4n + 6)$ is in $O(n)$

Question 2

1 / 1 pts

Asymptotically, $f_1(n) = 2n^2$ has a higher complexity than $f_2(n) = 10n$. However, f_1 is not larger than f_2 for every value of n. Beyond what value of n does f_1 overtake f_2?

[You can refer Section 1.3.3 of the textbook]

5

Question 3

1 / 1 pts

Google indexes hundreds of billions of pages [1]. Let us assume that the index contains one trillion pages. Performing a query is equivalent to searching through all the pages in the index.

A naive way to execute a query would be to search through all the pages. Imagine that Google has a state of the art server with a clock speed of 100 GHz. That is, it performs 100 billion machine instructions in one

second. Further assume that searching through one page on this server requires 20 machine instructions.

Given the above numbers, how many seconds would the server take to perform a query on Google's index?

[1] - How Search organizes information (https://www.google.com/search/howsearchworks/crawling-indexing/), accessed 18 Jan 2022

200

Question 4

1 / 1 pts

From your experience, is the above time the typical query time on Google?

- ○ Yes
- ◉ No

Question 5

1 / 1 pts

While doing asymptotic analysis, we ignore:

- ☑ lower order terms
- ☐ highest order term
- ☑ constant coefficients
- ☐ None of the above

Question 6

1 / 1 pts

Asymptotic analysis (Big O) tells us about:

- [ ] the exact time taken by the algorithm.
- [ ] the space utilization of the algorithm
- [x] the order of growth with respect to the problem size

Question 7

1 / 1 pts

We proved that the amortized cost of `resize()` is *O(m)* (as per Lemma 2.1) over all `add(i,x)` and `remove(i)` operations in an *ArrayStack* implementation of the *List* ADT. Which of the following statements expresses the same?

Mark all that apply.

- [ ] For n calls to add() or remove(), O(n) time has been spent in resize() so far.
- [ ] For n elements in the list, O(n) time has been spent in resize() so far.
- [ ] resize() takes O(n) time to execute. (where n is the number of elements)
- [ ] add() and remove() are O(n) operations because of resize().

Question 8

2 / 2 pts

Imagine a dynamic array that is doubled in every `resize()` operation (as we encountered in the lectures). How much time is spent when *n* `add()` and `remove()` operations are performed on it?

- (●) O(n)
- ( ) O(1)
- ( ) O(n^2)

Question 9

1 / 1 pts

Now, imagine a dynamic array that grows the backing array(new array) by a fixed amount in every `resize()` operation. How much time is spent in this array when *n* `add()` and `remove()` operations are performed on it?

- (●) n^2
- ( ) 1
- ( ) n

Question 10

3 / 3 pts

Consider the following *List* operations applied in the indicated order to an initially empty *ArrayDequeue* implementation with a backing array of size 2. For each operation, indicate the index in the backing array where insertion or removal occurs.

[Note: The implementation of *ArrayDequeue* is based on circular structure. append(x) is add(x) and pop(0) is remove(x) as per section 2.4 of the Textbook]

append(x)	0
append(x)	1
pop(0)	0
append(x)	0
pop(0)	1
append(x)	1
append(x)	2
append(x)	3
pop(0)	0
append(x)	0
append(x)	4

Answer 1:

0

Answer 2:

1

Answer 3:

0

Answer 4:

0

Answer 5:

1

Answer 6:

1

Answer 7:

2

Answer 8:

3

Answer 9:

0

Answer 10:

0

Answer 11:

4

Question 11

2 / 2 pts

In an *ArrayQueue* implementation, why is it necessary to "straighten" the *Queue* elements on `resize()`?

Mark all that apply.

- [ ] It is easier for the machine to maintain the elements this way.
- [x] The modulo formula for wraparound will yield incorrect results otherwise.

- [ ] So that space is used more efficiently.
- [ ] To reduce the runtime of subsequent operations.

Question 12

3 / 3 pts

Consider the following operations being performed in the given sequence on an initially empty deque. Show the state of deque after each operation:

Note: The state of deque will be shown as: [1,2,3,4] or [] if it is empty.

D.add_last(5)	[5]
D.add_first(3)	[3,5]
D.add_first(7)	[7,3,5]
D.first()	[7,3,5]
D.delete_last()	[7,3]
D.delete_last()	[7]
D.add_first(6)	[6,7]
D.last()	[6,7]
D.add_first(8)	[8,6,7]
D.last()	[8,6,7]

Answer 1:

[5]

Answer 2:

[3,5]

Answer 3:

[7,3,5]

Answer 4:

[7,3,5]

Answer 5:

[7,3]

Answer 6:

[7]

Answer 7:

[6,7]

Answer 8:

[6,7]

Answer 9:

[8,6,7]

Answer 10:

[8,6,7]

Question 13

1 / 1 pts

Resizing an array:

- [x] increases/decreases its capacity.
- [ ] changes the indices of existing elements in the array.
- [ ] does not retain the order of elements in the array.
- [ ] always creates a new and longer array.

Quiz score: **20** out of 20