Programming Assignment 4 - questions - Late/Resubmit

Graded

Student

Andrew Onozuka

Total Points

54 / 54 pts

Question 1

Big-O Justification **12** / 12 pts

1.1 **n + 5n^2 + 8n^4 is O(n^3)** **2** / 2 pts

✔ **– 0 pts** Correct: Does not hold, with justification

– 0 pts No deduction, but a reminder that you shouldn't pick n0 and C for the negative direction. n0 and C are used as part of saying the big-O relationship HOLDS, not that it doesn't.

– 1 pt Justification does not mention the larger growing term.

– 1 pt Incorrect answer with justification given

– 2 pts Incorrect answer with no justification or blank

1.2 **n! + n is O(n * log n)** **2** / 2 pts

✔ **– 0 pts** Correct: Does not hold, with justification

– 1 pt Justification does not mention the larger growing term.

– 1 pt Incorrect answer with justification given

– 2 pts Incorrect answer with no justification or blank

– 0 pts No deduction, but a reminder that you shouldn't pick n0 and C for the negative direction. n0 and C are used as part of saying the big-O relationship HOLDS, not that it doesn't.

1.3 **log n + n * log n + log(log n) is Ω(n)** **2** / 2 pts

✔ **– 0 pts** Correct: Holds, with justification that includes C, n0 and mention of larger-growing term

– 0.5 pts Justification contains minor error (e.g. incorrect/missing C, n0, lower/upper bound, etc.)

– 1 pt Incorrect answer with justification given

– 2 pts Incorrect answer with no justification or blank

1.4 **n^2 + n/4 + 6 is Θ(n^2)** **2** / 2 pts

✔ **– 0 pts** Correct: Does hold, with justification.

– 0.5 pts Justification contains minor error (e.g. incorrect/missing C, n0, lower/upper bound, etc.)

– 1 pt Justification does not mention big-Omega

– 1 pt Incorrect answer with some justification given

– 2 pts Incorrect answer with no justification or blank

1.5 **1/(n^50) + log32 is Ω(1)** **2** / 2 pts

✔ **– 0 pts** Correct: Holds, with justification that includes C, n0 and mention of larger-growing term

– 0 pts No deduction, but a reminder that you shouldn't pick n0 and C for the negative direction. n0 and C are used as part of saying the big-O relationship HOLDS, not that it doesn't.

– 1 pt Incorrect answer with justification given

– 2 pts Incorrect answer with no justification or blank

1.6 **1/(n^50) + log2 is O(1)** **2** / 2 pts

✔ **– 0 pts** Correct: Holds, with justification given including n0 and C, and rate of growth

– 0.5 pts Justification does not mention n0 and C, or does not mention rate of growth

– 1 pt Incorrect answer with some justification given

– 2 pts Incorrect answer with no justification or blank

Question 2

List Analysis **16** / 16 pts

2.1 **Did you read the format instructions?** **0** / 0 pts

✔ **+ 0 pts** Correct

+ 0 pts Incorrect

2.2 **Give a tight big-O bound for the best case running time of prepend in ArrayStringList** **2** / 2 pts

✔ **– 0 pts** Correct: Gives tight bound of n written as O or big-Theta, justifies by describing the array shifting down. May optionally mention expandCapacity.

– 0 pts No deduction, but a note that running expandCapacity would not change the bound in this case (since expandCapacity is O(n) as well)

– 1 pt Correct bound, but incorrect or insufficient explanation

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

2.3 **Give a tight big-O bound for the worst case running time of prepend in ArrayStringList** **2** / 2 pts

✔ **– 0 pts** Correct: Gives tight bound of n written as O or big-Theta, justifies by describing the array shifting down. May optionally mention expandCapacity.

– 0 pts No deduction, but a note that running expandCapacity would not change the bound in this case (since expandCapacity is O(n) as well)

– 1 pt Correct bound, but incorrect or insufficient explanation

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

2.4 **Give a tight big-O bound for the best case running time of prepend in LinkedStringList** **2** / 2 pts

– 1 pt Correct bound, but incorrect or insufficient explanation

✔ **– 0 pts** Correct: Gives tight bound of 1 or a constant written as O or big-Theta, justifies by describing that only a constant number of operations are needed.

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

2.5 **Give a tight big-O bound for the worst case running time of prepend in LinkedStringList** **2** / 2 pts

✔ **– 0 pts** Correct: Gives tight bound of 1 or a constant written as O or big-Theta, justifies by describing that only a constant number of operations are needed.

– 1 pt Correct bound, but incorrect or insufficient explanation

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

2.6 **Give a tight big-O bound for the best case running time of add in ArrayStringList** **2** / 2 pts

✔ **– 0 pts** Correct: Gives tight bound of 1 or a constant. Justified by expandCapacity NOT running.

– 1 pt Correct bound, but incorrect or insufficient explanation

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

2.7 **Give a tight big-O bound for the worst case running time of add in ArrayStringList** **2** / 2 pts

✔ **– 0 pts** Correct: Gives tight bound of n written as O or big-Theta. Justified by expandCapacity running in the worst case.

– 1 pt Correct bound, but incorrect or insufficient explanation

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

2.8 **Give a tight big-O bound for the best case running time of add in LinkedStringList** **2** / 2 pts

✔ **– 0 pts** Correct: Gives tight bound of n written as O or big-Theta, justifies by describing that all nodes must be looped over

– 1 pt Correct bound, but incorrect or insufficient explanation

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

2.9 **Give a tight big-O bound for the worst case running time of add in LinkedStringList** **2** / 2 pts

✔ **– 0 pts** Correct: Gives tight bound of n written as O or big-Theta, justifies by describing that all nodes must be looped over

– 1 pt Correct bound, but incorrect or insufficient explanation

– 1 pt Gives wrong bound with some justification

– 2 pts Blank or gives wrong bound with no justification

Question 3

Mystery Functions and Measuring — **26** / 26 pts

3.1 **Function Matching - Mystery A** — **1** / 1 pt

- ✔ **+ 1 pt** Correct
- **+ 0 pts** Incorrect

3.2 **Function Matching - Mystery B** — **1** / 1 pt

- ✔ **+ 1 pt** Correct
- **+ 0 pts** Incorrect

3.3 **Function Matching - Mystery C** — **1** / 1 pt

- ✔ **+ 1 pt** Correct
- **+ 0 pts** Incorrect

3.4 **Function Matching - Mystery D** — **1** / 1 pt

- ✔ **+ 1 pt** Correct
- **+ 0 pts** Incorrect

3.5 **Function Matching - Mystery E** — **1** / 1 pt

- ✔ **+ 1 pt** Correct
- **+ 0 pts** Incorrect

3.6 **Function Matching - Mystery F** — **1** / 1 pt

- ✔ **+ 1 pt** Correct
- **+ 0 pts** Incorrect

3.7 **Big-O bound for f1** — **2** / 2 pts

- ✔ **– 0 pts** Correct: O(n) or Θ(n), with some justification based on code
- **– 1 pt** Incorrect bound, with some justification based on code
- **– 1 pt** O(n) or Θ(n) with no justification based on code
- **– 2 pts** Incorrect bound with no justification, or blank

3.8 **Big-O bound for f2** **2** / 2 pts

✔ **– 0 pts** Correct: O(n) or Θ(n), with some justification based on code

– 1 pt Incorrect bound, with some justification based on code

– 1 pt O(n) or Θ(n) with no justification based on code

– 2 pts Incorrect bound with no justification, or blank

3.9 **Big-O bound for f3** **2** / 2 pts

✔ **– 0 pts** Correct: O(log(n)) or Θ(log(n)), with some justification based on code

– 1 pt Incorrect bound, with some justification based on code

– 1 pt O(log(n)) or Θ(log(n)) with no justification based on code

– 2 pts Incorrect bound with no justification, or blank

3.10 **Big-O bound for f4** **2** / 2 pts

✔ **– 0 pts** Correct: O(n^2) or Θ(n^2), with some justification based on code

– 1 pt Incorrect bound, with some justification based on code

– 1 pt O(n^2) or Θ(n^2) with no justification based on code

– 2 pts Incorrect bound with no justification, or blank

3.11 **Big-O bound for f5** **2** / 2 pts

✔ **– 0 pts** Correct: O(n^3) or Θ(n^3), with some justification based on code

– 1 pt Incorrect bound, with some justification based on code

– 1 pt O(n^3) or Θ(n^3) with no justification based on code

– 2 pts Incorrect bound with no justification, or blank

3.12 **Big-O bound for f6** **2** / 2 pts

✔ **– 0 pts** Correct: O(2^n) or Θ(2^n) or O(n*2^n), with some justification based on code

– 1 pt Incorrect bound, with some justification based on code

– 1 pt O(2^n) or Θ(2^n) with no justification based on code

– 2 pts Incorrect bound with no justification, or blank

3.13 **Description of measuring process** **2** / 2 pts

✔ **– 0 pts** Correct: Describes measuring process (mentioned System.nanoTime(), measuring duration, description of code, using a dummy call)

– 1 pt Does not describe measuring process

– 2 pts blank/missing

3.14 **Graphs** **6** / 6 pts

✔ **– 0 pts** Correct: Graph 1 shows a meaningful curve, either by a careful selection of range, or by comparing two different curves' measurements to one another

– 1 pt Graph 1 doesn't show a curve related to the method's actual behavior because of an ineffective range or other issue.

– 2 pts Missing graph 1

✔ **– 0 pts** Correct: Graph 2 shows a meaningful curve, either by a careful selection of range, or by comparing two different curves' measurements to one another

– 1 pt Graph 2 doesn't show a curve related to the method's actual behavior because of an ineffective range or other issue.

– 2 pts Missing graph 2

✔ **– 0 pts** Correct: Graph 3 shows a meaningful curve, either by a careful selection of range, or by comparing two different curves' measurements to one another

– 1 pt Graph 3 doesn't show a curve related to the method's actual behavior because of an ineffective range or other issue.

– 2 pts Missing graph 3

Question 4

Collaborators **0** / 0 pts

– 1 pt Nothing written in collaboration section (you are required to indicate whether or not you had collaborators)

– 0 pts "I did not collaborate with anyone" (No grade effect, just for staff use)

✔ **– 0 pts** Some collaborators listed (No grade effect, just for staff use)

Q1 Big-O Justification

12 Points

Indicate whether the following assertions are true or false, and give a justification.

If you are justifying the positive direction, give choices of n0 and C. For big-Θ, make sure to justify both big-O and big-Ω, or big-O in both directions.

If you are justifying the negative direction, indicate which term(s) can't work because one is guaranteed to grow faster or slower than the other.

Q1.1 n + 5n^2 + 8n^4 is O(n^3)

2 Points

- ○ True
- ◉ False

Justification

This is false. n + 5n^2 + 8n^4 is not O(n^3) because 8n^4 eventually grows faster than n^3 no matter what value you set for n0 or any positive constant c. If we set it so 8n^4 = c*n^3, for example c = 8, n = 1; we will be able to see that 8n^4 will grow larger faster than any c*n^3, because after n = 1, every point for 8n^4 is always larger than c*n^3. This means that any runtime equation with c*n^4 as the largest factor will be slower than any runtime equation with c*n^3 as the largest factor.

Q1.2 n! + n is O(n * log n)
2 Points

- ○ True
- ◉ False

Justification

This is false. n! + n is not O(n * log n) because n! will always grow faster than O(n * log n). This means that n! will always take more runtime and will be slower than O(n * log n). If we set c = 10, for example, the two equations equal each other at n = 1.545, 3.823. However after the second point, n! will always grow faster, therefore being the slower runtime equation. Because it is slower, the runtime equation cannot be O(n * log n).

Q1.3 log n + n * log n + log(log n) is Ω(n)
2 Points

- ◉ True
- ○ False

Justification

This is true. Because n is linear and log n + n * log n + log(log n) is not, no matter what the constant c, log n + n * log n + log(log n) will always grow larger, therefore having a longer runtime. Let's show this by setting c = 2, n = 94.622. We can see that after a certain n, log n + n * log n + log(log n) will inevitably grow larger than n. Because we can see that n will always be smaller than log n + n * log n + log(log n) past some n0 for any constant c, n is the lower bound. Therefore Ω(n) is true.

Q1.4 n^2 + n/4 + 6 is Θ(n^2)
2 Points

- (●) True
- ( ) False

Justification

This is true. n^2 + n/4 + 6 is Θ(n^2) because the upper bound is O(n^2) and the lower bound is Ω(n^2). This means that the tight bound Θ(n^2) is true, since a tight bound is true when it abides by both the lower and upper bounds. We can see that O(n^2) is true because n^2 + n/4 + 6 is always smaller than or equal to c*n^2 for any n0 and any c >= 1. We can see that Ω(n^2) is also true if we just flip the previous rule, so n^2 + n/4 + 6 will always be greater than or equal to n^2 for any n0 and constant c <= 1. Because both the upper and lower bounds for n^2 + n/4 + 6 is n^2, then the statement n^2 + n/4 + 6 is Θ(n^2) must also be true.

Q1.5 1/(n^50) + log32 is Ω(1)
2 Points

- (●) True
- ( ) False

Justification

This is true. 1/(n^50) + log32 is Ω(1) because for any constant c and n0, 1/(n^50) + log32 >= c*1. If we set c = 1 and n = 0 (although you could set any n as n0 in this case), we will see that log 32 > 1 always.

Q1.6 1/(n^50) + log2 is O(1)
2 Points

- (●) True
- ( ) False

Justification

This is true. 1/(n^50) + log2 is O(1) because for any constant c and n0, 1/(n^50) + log2 <= c*1. If we set c = 1 and n0 = 2, we can see that 1/(n^50) + log2 <= 1. Because we can see that O(1) is always bigger than 1/(n^50) + log2, it means that O(1) is indeed the upper bound, making the statement true.

Q2 List Analysis
16 Points

Answer the following questions, and justify them with one or two sentences. Unless specified otherwise, all runtime is the worst-case runtime.

Format Instructions. Please read the following instructions carefully.

For your runtime, do not write Θ(), only what is inside of ().
Only give the most relevant terms. For example, if you arrived at the answer Θ(4n + 2), then this would simplify to Θ(n). Your answer would just be n.
For answers with exponential terms, you should format it using ^. For example, if your answer is $\Theta(n^2)$, your answer would be n^2.
For answers that involve log, you should explicitly write "log" with parentheses. For example, if your answer is Θ(log(n)), your answer would be log(n). You don't have to write the base of the log.
For answers with multiple terms, do not include spaces. For example, if your answer is Θ(n*m), your answer would be m*n. Additionally, order them alphabetically.

Q2.1 Did you read the format instructions?
0 Points

◉ I have read and understood the format requirements

Q2.2 Give a tight big-O bound for the best case running time of prepend in ArrayStringList
2 Points

Big O bound

n

Justification

The tight big-O bound for the best case running time of prepend in ArrayStringList is n because it always runs expandCapacity when prepending, which runs n times in order to copy all of the previous elements to the expanded array. Then in order to insert, every element is shifted an index which also runs n times. There are no circumstances where prepend does not call these methods, therefore the best case run time is Θ(n).

Q2.3 Give a tight big-O bound for the worst case running time of prepend in ArrayStringList
2 Points

Big O bound

n

Justification

The tight big-O bound for the worst case running time of prepend in ArrayStringList is also n because it always runs expandCapacity when prepending, which runs n times in order to copy all of the previous elements to the expanded array. Then in order to insert, every element is shifted an index which also runs n times. There are no circumstances where prepend does not call these methods, therefore the worst case run time is also Θ(n).

Q2.4 Give a tight big-O bound for the best case running time of prepend in LinkedStringList
2 Points

Big O bound

1

Justification

The tight big-O bound for the best case running time of prepend in LinkedStringList is 1 because to prepend, it is constant time. This is because the method does not depend on the size of the list, and therefore the best case is Θ(1).

Q2.5 Give a tight big-O bound for the worst case running time of prepend in LinkedStringList
2 Points

Big O bound

1

Justification

The tight big-O bound for the worst case running time of prepend in LinkedStringList is also 1. Because the prepend method always makes 3 evaluations, the runtime is constant and does not depend on n. Therefore, the worst case is also Θ(1).

Q2.6 Give a tight big-O bound for the best case running time of add in ArrayStringList
2 Points

Big O bound

1

Justification

The tight big-O bound for the best case running time of add in ArrayStringList is 1. This is because when adding to the end on an array where the number of elements is less than the size of the array (i.e. there are still spots left in the array to fill with values), the expandCapacity method will not copy into a new array and simply return. Then since adding to the end only has 2 evaluations, the best case is constant time and Θ(1).

Q2.7 Give a tight big-O bound for the worst case running time of add in ArrayStringList
2 Points

Big O bound

n

Justification

The tight big-O bound for the worst case running time of add in ArrayStringList is n. This is because if the number of elements equals the size of the current array, expandCapacity runs when add is called. Because there are no more free spots and expandCapacity must run, it loops through the array and copies its contents into the new extended array, which runs n times. Because n is the largest term in the runtime, the tight big-O bound on the worst case run time is Θ(n).

Q2.8 Give a tight big-O bound for the best case running time of add in LinkedStringList
2 Points

Big O bound

n

Justification

The tight big-O bound for the best case running time of add in LinkedStringList is n. This is because whenever we add in LinkedStringList, it must loop n times to the end of the LinkedList in order to add at the end. Because of this requirement to loop to the end of the list, the tight big-O bound for the best case runtime is Θ(n).

Q2.9 Give a tight big-O bound for the worst case running time of add in LinkedStringList
2 Points

Big O bound

n

Justification

The tight big-O bound for the worst case running time of add in LinkedStringList is also n. This is because whenever we add in LinkedStringList, it must loop n times to the end of the LinkedList in order to add at the end. Because of this requirement to loop to the end of the list, the tight big-O bound for the worst case runtime is also Θ(n).

Q3 Mystery Functions and Measuring

26 Points

Determine which mystery method A-F corresponds to the implementations above by measuring against provided (but hidden) implementation.

Determine the tightest big-O bound for each function, and justify it with a few sentences. Give only the most relevant term, so use, for example *O(n)*, not *O(4n + 2)*.

Describe how you measured your functions.

Submit 3 relevant graphs that display your measurements.

Q3.1 Function Matching - Mystery A

1 Point

- ○ f1
- ◉ f2
- ○ f3
- ○ f4
- ○ f5
- ○ f6

Q3.2 Function Matching - Mystery B

1 Point

- ◉ f1
- ○ f2
- ○ f3
- ○ f4
- ○ f5
- ○ f6

Q3.3 Function Matching - Mystery C
1 Point

- ○ f1
- ○ f2
- ○ f3
- ○ f4
- ◉ f5
- ○ f6

Q3.4 Function Matching - Mystery D
1 Point

- ○ f1
- ○ f2
- ○ f3
- ◉ f4
- ○ f5
- ○ f6

Q3.5 Function Matching - Mystery E
1 Point

- ○ f1
- ○ f2
- ○ f3
- ○ f4
- ○ f5
- ◉ f6

Q3.6 Function Matching - Mystery F
1 Point

- ○ f1
- ○ f2
- ◉ f3
- ○ f4
- ○ f5
- ○ f6

Q3.7 Big-O bound for f1
2 Points

Big O bound

n

Justification

The big-O bound for f1 is n because of a loop in f1 that runs n times. The runtime equation is 3n + 2, but since we do not care about the constants when expressing runtime, the big-O bound is n, therefore O(n).

Q3.8 Big-O bound for f2
2 Points

Big O bound

n

Justification

The big-O bound for f2 is also n. The loop in the f2 method is n/3 times, but because n/3 < n, the upper bound is n. Therefore big-O bound is O(n).

Q3.9 Big-O bound for f3
2 Points

Big O bound

log(n)

Justification

The big-O bound for f3 is log(n). For the f3 method, the outside for-loop looks as if it would run n times, but due to the inner for-loop, it actually only runs log(n) times. This is because the inner for-loop continues to change the value of n, thus making the runtime more efficient than a simple look at the outer for-loop may suggest. After the loop is iterated through log(n) times, the while loop also finishes, therefore the big-O bound for f3 is log(n).

Q3.10 Big-O bound for f4
2 Points

Big O bound

n^2

Justification

The big-O bound for f4 is n^2 because of a nested loop. The outer for loop is iterated through n times and the inner loop is n - i iterations, but since for big-O we want the most dominant term only, we can essentially disregard all of the - i terms. This leaves the largest term of n^2, which is the big-O bound for f4, which is O(n^2).

Q3.11 Big-O bound for f5
2 Points

Big O bound

n^3

Justification

The big-O bound for f5 is n^3 because the outer loop has a runtime of n^2 and the inner loop has a run time of n. Because n^2 * n = n^3, the big-O bound for f5 is O(n^3).

Q3.12 Big-O bound for f6
2 Points

Big O bound

n*2^n

Justification

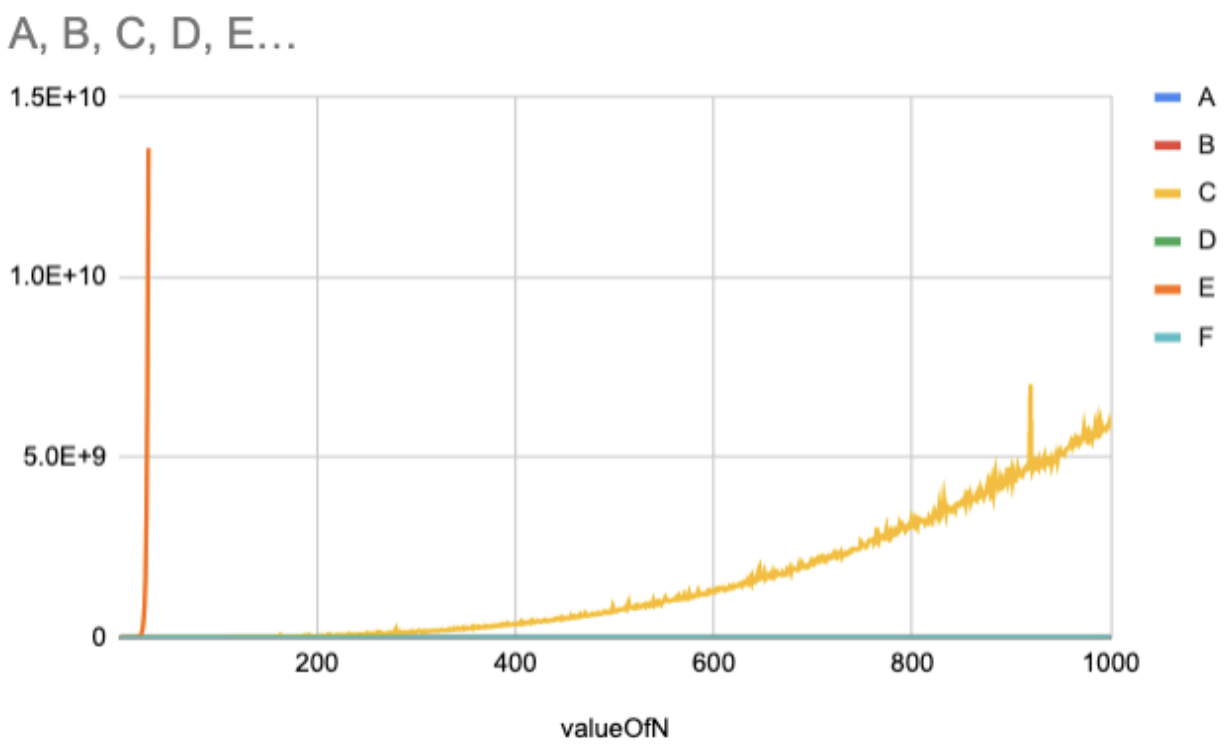
The big-O bound for f6 is n*2^n because the inside loop iterates through 2^n times. This is multiplied by the outer loop which iterates through n times, therefore the big-O bound is O(n*2^n).

Q3.13 Description of measuring process
2 Points

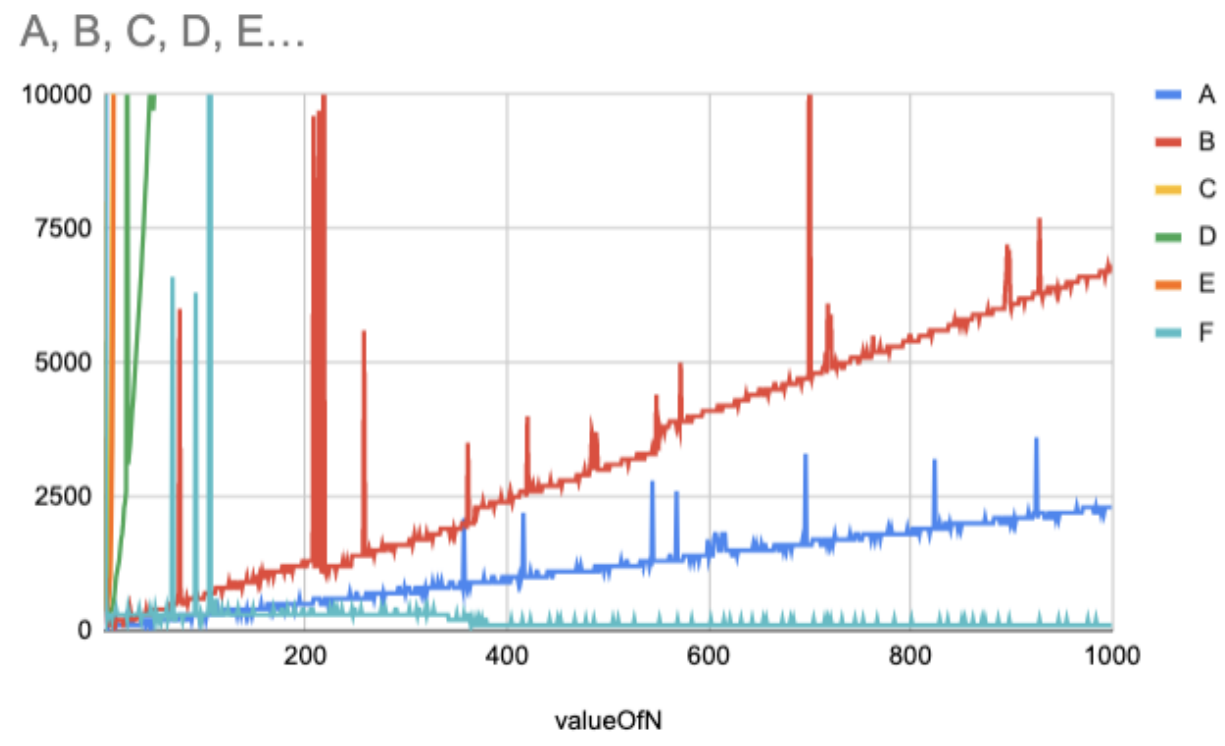
To measure, you first calculate the runtime and get the big-O bounds for each method. Using System.nanoseconds(), you are able to see the time it takes for each function to run. Doing this for n values from 0 to 1000 with the exception of E (as n*2^n is far too inefficient), I took the console values and brought the comma separated values into a spreadsheet to graph. We set valueOfN to the x axis and then we can see the runtimes visualized.

Q3.14 Graphs
6 Points

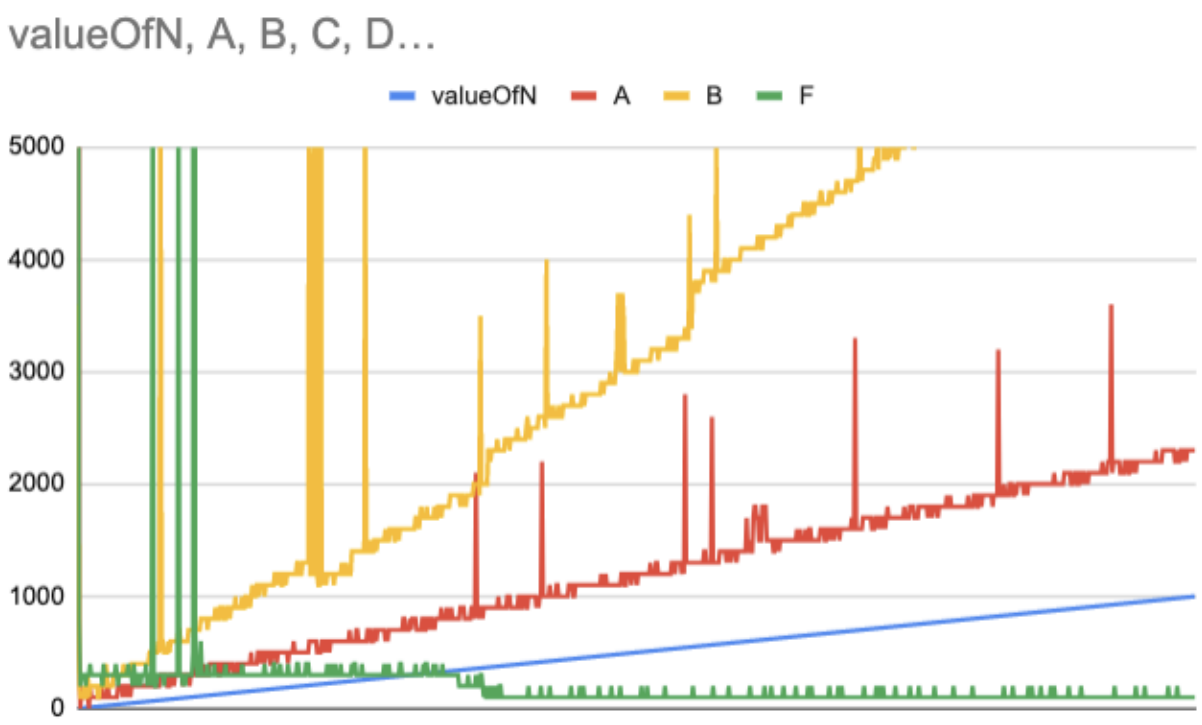
1.png Download



2.png Download



3.png Download



Q4 Collaborators
0 Points

You are required to indicate whether or not you had collaborators on this assignment. If you did not have collaborators, type “I did not collaborate with anyone". Otherwise list the name of your collaborators.

Benjamin Liang