CSC108H Winter 2024 Worksheet 32 : Run Time Complexity of Functions

In this exercise, you will explore how to read code and figure out how efficient that code is. For each of the following functions, indicate the runtime with respect to the given quantity.

1. `L` refers to a list. If there are k items in the list, roughly how many iterations are there?

```
sum = 0
for value in L:
    sum = sum + value
```

(a) 1 (b) 2 (c) 4 (d) $k/2$ (e) k (f) k^2

2. `L` refers to a list. If there are k items in the list, roughly how many iterations are there?

```
sum = 0
for i in range(len(L)):
    sum = sum + L[i]
```

(a) 1 (b) 2 (c) 4 (d) $k/2$ (e) k (f) k^2

3. `L` refers to a list. If there are k items in the list, roughly how many iterations are there?

```
sum = 0
i = 0
while i < len(L):
    sum = sum + L[i]
    i = i + 1
```

(a) 1 (b) 2 (c) 4 (d) $k/2$ (e) k (f) k^2

4. `L` refers to a list. If there are k items in the list, roughly how many iterations are there?

```
sum = 0
i = 0
while i < len(L):
    sum = sum + L[i]
    i = i + 2
```

(a) 1 (b) 2 (c) 4 (d) $k/2$ (e) k (f) k^2

5. `L` refers to a list. If there are k items in the list, roughly how many iterations are there?

```
# Precondition: len(L) % 4 == 0
sum = 0
i = 0
while i < len(L):
    sum = sum + L[i]
    i = i + len(L) // 4
```

(a) 1 (b) 2 (c) 4 (d) $k/2$ (e) k (f) k^2

6. The first five questions all involve looping over a list. If the length of the list is doubled, some of the loops will take twice as many iterations as before.[1] Which questions contain these loops? Circle all that apply.

(a) question 1 (b) question 2 (c) question 3 (d) question 4 (e) question 5

7. The first five questions all involve looping over a list. If the length of the list is doubled, some of the loops will take exactly the same number of iterations as before.[2] Which questions contain these loops? Circle all that apply.

(a) question 1 (b) question 2 (c) question 3 (d) question 4 (e) question 5

[1] This is *linear running time*: the running time is proportional to the size of the input.
[2] This is *constant running time*: the running time does not depend on the size of the input.

CSC108H Winter 2024 Worksheet 32 : Run Time Complexity of Functions

8. `L` refers to a list. If there are k items in the list, roughly how many times is the assignment statement `sum = val1 - val2` executed?

```
sum = 0
for val1 in L:
    for val2 in L:
        sum = val1 - val2
```

(a) 1 (b) 4 (c) $k/2$ (d) k (e) $10k$ (f) k^2

9. Roughly how many times is the assignment statement `sum = sum + i` executed?

```
sum = 0
for i in range(k * k):
    sum = sum + i
```

(a) 1 (b) 4 (c) $k/2$ (d) k (e) $10k$ (f) k^2

10. `L` refers to a list. If there are k items in the list, roughly how many times is the assignment statement `sum = sum + i + j` executed?

```
sum = 0
for i in range(10):
    for j in range(len(L)):
        sum = sum + i + j
```

(a) 1 (b) 4 (c) $k/2$ (d) k (e) $10k$ (f) k^2

11. Which of questions 8, 9, and 10 contain code that has linear running time? Circle all that apply.

(a) question 8 (b) question 9 (c) question 10

12. *Quadratic* running time is proportional to the square of the size of the input. Which of questions 8, 9, and 10 contain code that has quadratic running time? Circle all that apply.

(a) question 8 (b) question 9 (c) question 10

CSC108H Winter 2024 Worksheet 33 : Sorting Algorithms

1. The list `[4, 2, 5, 6, 7, 3, 1]` is shown below after each pass of a sorting algorithm:

```
[1, 2, 5, 6, 7, 3, 4]
[1, 2, 5, 6, 7, 3, 4]
[1, 2, 3, 6, 7, 5, 4]
[1, 2, 3, 4, 7, 5, 6]
[1, 2, 3, 4, 5, 7, 6]
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
```

   Which sorting algorithm is being executed?

   (a) bubble sort

   (b) selection sort

   (c) insertion sort

2. The list `[4, 2, 5, 6, 7, 3, 1]` is shown below after each pass of a sorting algorithm:

```
[2, 4, 5, 6, 3, 1, 7]
[2, 4, 5, 3, 1, 6, 7]
[2, 4, 3, 1, 5, 6, 7]
[2, 3, 1, 4, 5, 6, 7]
[2, 1, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
```

   Which sorting algorithm is being executed?

   (a) bubble sort

   (b) selection sort

   (c) insertion sort

3. The list `[4, 2, 5, 6, 7, 3, 1]` is shown below after each pass of a sorting algorithm:

```
[4, 2, 5, 6, 7, 3, 1]
[2, 4, 5, 6, 7, 3, 1]
[2, 4, 5, 6, 7, 3, 1]
[2, 4, 5, 6, 7, 3, 1]
[2, 4, 5, 6, 7, 3, 1]
[2, 3, 4, 5, 6, 7, 1]
[1, 2, 3, 4, 5, 6, 7]
```

   Which sorting algorithm is being executed?

   (a) bubble sort

   (b) selection sort

   (c) insertion sort

CSC108H Winter 2024 Worksheet 33 : Sorting Algorithms

4. List `[8, 2, 8, 7, 3, 1, 2]` is being sorted using bubble sort. Fill in the blanks to show the list after each pass.

   After the 1st pass: `[2, 8, 7, 3, 1, 2, 8]`

   After the 2nd pass: `[2, 7, 3, 1, 2, 8, 8]`

   After the 3rd pass: `[2, 3, 1, 2, 7, 8, 8]`

   After the 4th pass: _____

   After the 5th pass: _____

   After the 6th pass: _____

5. List `[1, 5, 8, 7, 6, 1, 7]` is being sorted using selection sort. Fill in the blanks to show the list after each pass.

   After the 1st pass: `[1, 5, 8, 7, 6, 1, 7]`

   After the 2nd pass: `[1, 1, 8, 7, 6, 5, 7]`

   After the 3rd pass: `[1, 1, 5, 7, 6, 8, 7]`

   After the 4th pass: _____

   After the 5th pass: _____

   After the 6th pass: _____

   After the 7th pass: _____

6. List `[6, 8, 2, 1, 1, 9, 4]` is being sorted using insertion sort. Fill in the blanks to show the list after each pass.

   After the 1st pass: `[6, 8, 2, 1, 1, 9, 4]`

   After the 2nd pass: `[6, 8, 2, 1, 1, 9, 4]`

   After the 3rd pass: `[2, 6, 8, 1, 1, 9, 4]`

   After the 4th pass: _____

   After the 5th pass: _____

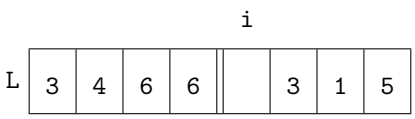
   After the 6th pass: _____

   After the 7th pass: _____

CSC108H Winter 2024 Worksheet 34 : Insertion Sort Analysis

1. **Insertion Sort: Worst Case**

   (a) In the list below, 4 passes of the insertion sort algorithm have been completed, and the double bar separates the sorted part of the list from the unsorted part. The item at index `i` is missing. Fill in the missing item with a value that will cause `insert(L, i)` to perform the most number of steps. (As a reminder, this is called the *worst case*.)



   (b) When `insert(L, i)` is executed on the example list, how many times does the while loop iterate?

   (c) In general, in the *worst* case, on the pass of insertion sort that calls `insert(L, i)`, how many times does the while loop iterate? (Your answer should be a formula that involves `i`.)

   (d) Does the code inside the loop body (i.e. the two assignment statements) have constant running time, linear running time, quadratic running time, or some other running time?

   (a) constant (b) linear (c) quadratic (d) something else

   (e) In terms of `i`, in the *worst* case, does function `insert` have constant running time, linear running time, quadratic running time, or some other running time?

   (a) constant (b) linear (c) quadratic (d) something else

   (f) In function `insertion_sort`, the first time that function `insert` is called, `i` is `0`; the second time, `i` is `1`; and so on. What value does `i` have the last time that function `insert` is called?

   (g) For the call `insertion_sort(L)`, in the *worst* case, write a formula expressing how many **total loop iterations** are made during all the calls to `insert`.

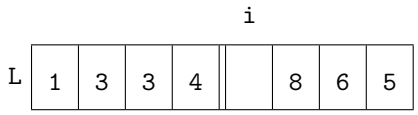
   (h) In the *worst* case, does `insertion_sort` have constant running time, linear running time, quadratic running time, or some other running time?

   (a) constant (b) linear (c) quadratic (d) something else

CSC108H Winter 2024 Worksheet 34 : Insertion Sort Analysis

2. **Insertion Sort: Best Case**

(a) In the list below, 4 passes of the insertion sort algorithm have been completed, and the double bar separates the sorted part of the list from the unsorted part. The item at index `i` is missing. Fill in the missing item with a value that will cause `insert(L, i)` to perform the *fewest* number of steps. (That's called the *best case*).



(b) When `insert(L, i)` is executed on the example list, how many times does the while loop iterate?

(c) In general, in the *best* case, on the pass of insertion sort that calls `insert(L, i)`, how many times does the while loop iterate?

(d) Other than the loop, we have some assignment statements and comparisons (in the loop condition) that are done within `insert(L, i)`. When called on the example list, how many **assignment statements + comparisons** are made?

(e) In general, in the *best* case, on the pass of insertion sort that calls `insert(L, i)`, how many **assignment statements + comparisons** are made?

(f) In the *best* case, does `insert` have constant running time, linear running time, quadratic running time, or some other running time?

(a) constant (b) linear (c) quadratic (d) something else

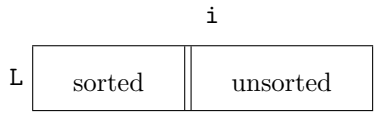
(g) For the *best* case, write a formula expressing how many **total assignment statements + comparisons** are made during all the calls to `insert`.

(h) In the *best* case, does `insertion_sort` have constant running time, linear running time, quadratic running time, or some other running time?

(a) constant (b) linear (c) quadratic (d) something else

CSC108H Winter 2024 Worksheet 35 : Selection Sort Analysis

1. In the list below, `i` passes of the selection sort algorithm have been completed, and the double bar separates the sorted part of the list from the unsorted part.



   (a) `get_index_of_smallest(L, i)` works by comparing pairs of items from the unsorted section. If there are `n` items in `L`, when `get_index_of_smallest(L, i)` is executed, how many pairs of items are **compared**? (Your answer should be a formula involving `n` and `i`.)

   (b) For function `get_index_of_smallest(L, i)`, is there a worst case and a best case?

   (c) In terms of the number of items in the unsorted section, does `get_index_of_smallest` have constant running time, linear running time, quadratic running time, or some other running time?

   (a) constant (b) linear (c) quadratic (d) something else

   (d) In function `selection_sort`, the first time that function `get_index_of_smallest` is called, `i` is `0`; the second time, `i` is `1`; and so on. What value does `i` have the last time that function `get_index_of_smallest` is called?

   (e) For the call `selection_sort(L)`, write a formula expressing how many **comparisons** are made during all the calls to `get_index_of_smallest`.

   (f) In terms of the length of the list, does `selection_sort` have constant running time, linear running time, quadratic running time, or some other running time?

   (a) constant (b) linear (c) quadratic (d) something else