CSC108H Worksheet: Analysis of Sorting Algorithms

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*.)

					i			
L	3	4	6	6		3	1	5

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

(c) When `insert(L, i)` is called on the example list, how many assignment statements are executed?

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

(e) In general, in the *worst* case, on pass `i` of insertion sort, how many assignment statements are executed? (Again, your answer should be a function that involves `i`.)

(f) 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

(g) 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?

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

(i) 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 Worksheet: Analysis of Sorting Algorithms

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*).

					i			
L	1	3	3	4		8	6	5

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

   (c) When `insert(L, i)` is called on the example list, how many assignment statements are executed?

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

   (e) In general, in the *best* case, on pass `i` of insertion sort, how many assignment statements are executed?

   (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

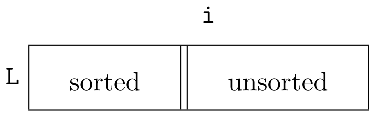
   (g) For the *best* case, write a formula expressing how many comparisons are made during all the calls to `insert`.

   (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

3. **Selection Sort**

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 function 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`.

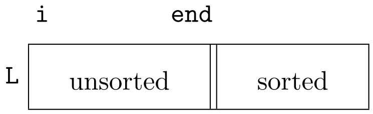
(f) In terms of the length of the list, does `selection_sort` have constant running time, linear running time, quadratic running time, or some other running time?

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

CSC108H Worksheet: Analysis of Sorting Algorithms

4. Bubble Sort

If the list below contains `n` items, `n - end - 1` passes of the bubble sort algorithm have been completed, and the double bar separates the sorted part of the list from the unsorted part.



(a) If there are `k` items in the unsorted part of `L`, when the inner loop is executed, how many pairs of items are compared?

(b) In the *worst* case, if there are `k` items in the unsorted part of `L`, when the inner loop is executed, how many assignment statements are executed? Count a swap as two assignment statements.

(c) In terms of `k`, does the inner loop 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 `bubble_sort`, in terms of `n`, how many pairs of items are compared during the first iteration of the outer loop?

(e) In function `bubble_sort`, how many pairs of items are compared during the last iteration of the outer loop? Write your answer in terms of `n`.

(f) For the call `bubble_sort(L)`, write a formula in terms of `n` expressing how many pairs of items are compared during all the iterations of the inner loop.

(g) For the call `bubble_sort(L)`, in the *best* case, how many assignment statements are executed?

(h) For the call `bubble_sort(L)`, in the *worst* case, how many assignment statements are executed?

(i) Does `bubble_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