CSC317:Data Structures and Algorithm Design

Homework 1

Aaron Jesus Valdes

February 21, 2021

Problem 1

a

Between n=2 and n=588 bubble sort beats heap sort.

b

$$512n\log_2(n) \leq 8n^2 \quad \text{Solve for n} \tag{1}$$

$$512n\log_2(n) - 8n^2 = 0 \tag{2}$$

$$8n(64\log_2(n) - n) = 0 \tag{3}$$

$$64\log_2(n) - n = 0 \quad \text{n=0 is a solution} \tag{4}$$

$$\log_2(n) = \frac{n}{64} \tag{5}$$

$$2^{\log_2(n)} = 2^{\frac{n}{64}} \tag{6}$$

$$n = 2^{\frac{n}{64}} \quad \text{Convert to Lambert form} \tag{7}$$

$$n(2^{-\frac{n}{64}}) = (2^{\frac{n}{64}})(2^{-\frac{n}{64}}) \tag{8}$$

$$n((e^{\ln(2)})^{-\frac{n}{64}}) = 1 \tag{9}$$

$$n(e^{-\frac{n\ln(2)}{64}}) = 1 \quad u = -\frac{n\ln(2)}{64} n = -\frac{64u}{ln(2)} \tag{10}$$

$$(-\frac{64u}{ln(2)})(e^u) = 1 \tag{11}$$

$$ue^u = -\frac{\ln(2)}{64} \quad \text{Lambert form} \tag{12}$$

$$u = W_{-1}(-\frac{\ln(2)}{64}) \qquad u = W_0(-\frac{\ln(2)}{64}) \quad \text{Substitute u back} \tag{13}$$

$$-\frac{n\ln(2)}{64} = W_{-1}(-\frac{\ln(2)}{64}) \qquad -\frac{n\ln(2)}{64} = W_0(-\frac{\ln(2)}{64}) \quad \text{Substitute u back} \tag{14}$$

$$n = \frac{-64W_0(-\frac{\ln(2)}{64})}{ln(2)} \qquad n = \frac{-64W_{-1}(-\frac{\ln(2)}{64})}{ln(2)} \quad \text{We need to evaluate} \tag{15}$$

(16)

Therefore the solution are n≥1.011 or n≤588.92, however n=1.011 cannot be a solution since the minimum amount of numbers we need to have to sort is n=2 therefore the only solution is n≤588.92
This solution is a fraction which means that we have to take the floor of it to be able to have an integer solution ∴ n≤588

Problem 2

What is the worst-case time complexity of the algorithm SEARCH-A?(Linked List)

The worst case scenario does not depend if the set of integers S_i is sorted or unsorted given the fact that the time complexity does not change between the two sets. If the set of integers is unsorted or sorted,we can perform linear search whose worst-case time complexity is O(n) given the fact that the time complexity to traverse through a linked-list is O(n).

What is the worst-case time complexity of the algorithm SEARCH-B?(Queue)

The worst case scenario does not depend if the set of integers S_i is sorted or unsorted given the fact that the time complexity does not change between the two sets. If the set of integers is unsorted or sorted,we can perform linear search whose worst-case time complexity is O(n) given the fact that the time complexity to traverse through a queue is O(n).

What is the worst-case time complexity of the algorithm SEARCH-C?Array

The worst case scenario depends if the set of integers S_i is sorted or unsorted given the fact that the time complexity drastically changes between the two.
If the set of integers is sorted we can use binary-search whose worst-case time complexity is O(log(n)).
However, if the set of integers in unsorted we can use linear-search whose worst-case time complexity is O(n).

Problem 3

a

The lower bound for a problem is the worst case running time for the best possible algorithm for that problem. Therefore for the sorting algorithm, the best possible algorithm is the merge sort whose worst-case running time is $O(nlog(n))$ meaning that the lower-bound for the sorting problem is $\Omega(nlog(n))$

b

1. Bubble Sort

   Worst Case: O(n^2)

2. Insertion Sort

   Worst Case: O(n^2)

3. Selection Sort

   Worst Case: O(n^2)

4. Quicksort

   Worst Case: O(n^2)

5. Mergesort

   Worst Case: $O(nlog(n))$

c

The upper bound for the sorting algorithm is $O(n^2)$

d

From the previous list there is only one that is optimal which is the merge sort since it satisfies the conditions to be an optimal algorithm since the upper bound is the same as the lower bound.

1 Problem 4

Algorithm 1 Insertion-Sort(A) non-decreasing order

```
1: for j ← 2 to length[A] do
2:     key ← A[j]
3:     i ← j − 1
4:     while i > 0 and A[i] > key do
5:         A[i + 1] ← A[i]
6:         i ← i − 1
7:     A[i + 1] ← key
```

Algorithm 2 Insertion-Sort(A) non-increasing order

```
1: for j ← 2 to length[A] do
2:     key ← A[j]
3:     i ← j − 1
4:     while i > 0 and A[i] < key do          ▷ We just need to change the sign
5:         A[i + 1] ← A[i]
6:         i ← i − 1
7:     A[i + 1] ← key
```

Problem 5

Loop invariant: For every iteration of the 'for' loop , the sub-array A[1 ... j-1] is going to consist of the original element that are in a sorted order

Initialization: Prior to the first iteration we have j=2 therefore the sub-array A[i ... j-1] is going to be equal to A[1] which means that the loop invariant holds true since an array of a single element is always sorted.

Maintenance: Given the fact that the while loop places A[j] in the correct position for each iteration, it means that the sub-array A[1 ... j] becomes sorted. Therefore at the beginning of the next iteration where j becomes j+1, the loop invariant holds true since the subarray A[i ... j] is sorted.

Termination: The outer loop terminates when $j > n$ which means that it terminates when j=n+1. Therefore the sub-array A[1 ... j-1] becomes A[1 ...n], which means that by the definition of the loop invariant A[1 ...n] must be in a sorted order which is true and therefore the termination property holds true.