COMP 5511 / Fall 2020 Section DD BIPIN C DESAI

Assignment 3

Group 9

Thi Ngoc Hop NGUYEN, ID 29729283, thing_n@encs.concordia.ca

Weiwei XIAO, ID 40069298, we_iao@encs.concordia.ca

Adnan ALI, ID 40181614, al_adnan@encs.concordia.ca

Patrick DRUMMOND, ID 40185198, p_drummo@encs.concordia.ca

Question 1

As described in the notes, insertion sort goes sequentially through the array when making comparisons to find the proper place for the element that is currently being processed. Suppose that the sequential search is replaced by a binary search.

a) Will the change decrease the overall cost of insertion sort? Explain.

We consider the overall cost as the number of comparisons plus the number of swaps, when replacing the sequential search by a binary search, we can consider 3 cases:

* **The best case: With the sequential search, the insertion sort costs 0 swap and (n-1) comparisons. With the binary search, the the insertion sort costs 0 swap and ($n \log_2 n$) comparisons. Therefore, for the best case, the change from sequential search to binary search doesn't decrease the cost of insertion sort (it increases the cost instead)**

* **The worst case: With the sequential search, the insertion sort costs ($n^2/2$) swaps and ($n^2/2$) comparisons. The binary search helps decrease the number of comparisons from ($n^2/2$) to ($n \log_2 n$) and maintain the number of swap. Therefore, in the worst case, this change decreases the cost of insertion sort.**

* **The average case: With the sequential search, the insertion sort costs ($n^2/4$) swaps and ($n^2/4$) comparisons. With the binary search, the number of swaps is the same, the number of comparisons is ($n \log_2 n$), which is less than ($n^2/4$) when n is large. Therefore, this change decreases the cost of insertion sort in the average case.**

b) With respect to overall cost, will the modified version of insertion source be in a faster θ-category than the version in the notes? Explain.

As explained in the answer 1a, the change from sequential search to binary search decreases the number of comparisons, hence decreases the overall cost of insertion sort.

However, the cost of inserting elements at their right positions (i.e. the number of swaps) doesn't change (it's ($n^2/4$)). Therefore, θ-category is the same, it's θ(n^2)

Question 2
Consider applying radix sort to a list of 10,000 integers in the range from 1,000,000 to 9,999,999

a) Not counting the space needed to store the integers themselves, how many bytes of memory would be required? Assume pointers occupy 4 bytes.

We know that when using radix sort with a linked list for each bin, each integer associates with 1 pointer. For 10,000 integers, we need 10,000 pointers.
Besides, each bin needs 1 pointer to point to its linked list. We have 10 bins (represent for 10 digits), so we need 10 more pointers.
Furthermore, when we starts processing $(k+1)^{th}$ digit after finishing with k^{th} digit, we need another 10 bins to contain the pending list, thus we need another 10 more pointers.
All in all, we need 10,000 + 10 + 10 = 10,020 pointers.
Each pointer occupies 4 bytes, therefore it requires 10,020 * 4 = 40,080 bytes of memory

b) How many times would radix sort examine each value in the list?

Each integer in the list has 7 digits. The radix sort needs to examines all of these 7 digits. Hence, it will examine each value 7 times.