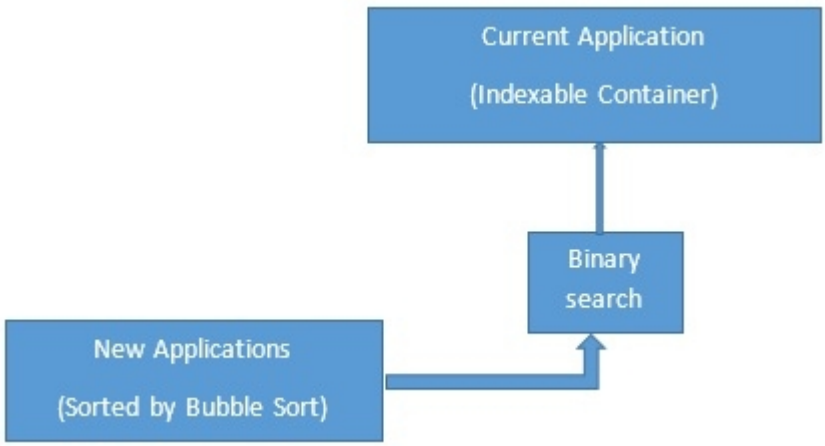
11/20/2016

Group Activity 4: Sorting

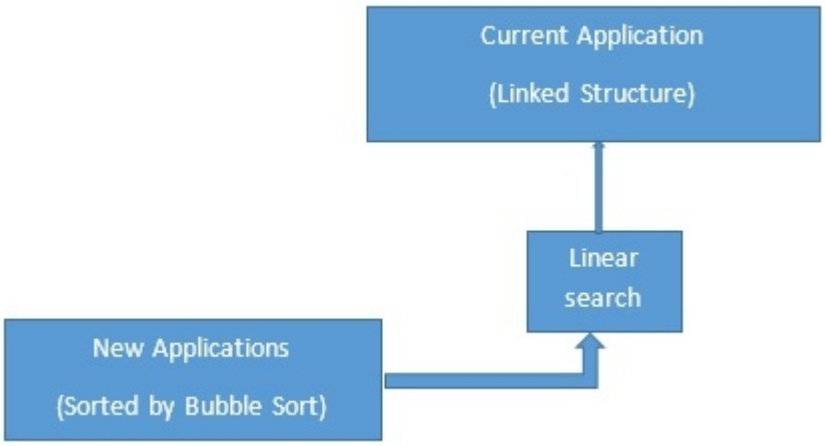
Authors:
Evan Furbeyre, Timothy Fye, Eli Goodwin, Vadiwoo Karuppiah, Robert Newton, Kevin Wong

1) Describe an algorithm to sort the new come in applications and include them into the larger stack.

Assuming an indexable container is being used to store the current applications, the new applicant container would be sorted using a bubble sort. Once the new applicant container is sorted, a binary search would be used on the current applicant container to find the correct index for inserting the new applicant into the current applicant container. Finally, once a new application has been inserted, its index would serve as the new starting pointing for the binary search. This is because no element in the new applicant container could be ordered before it or in the current applicant container. This would continue until the new applicant container was emptied.



If a linked structure was being used to store applicants new and current, the new applicant list would be sorted using a bubble sort. Once the new applicant list was sorted, a linear search would be used to locate the appropriate location for insertion in the current applicant list. Finally, once the new applicant was inserted, the linear search would continue from that location. It would search for the next location to insert a new applicant into the current application list and this would continue until the new applicant list was empty.



2) Describe an algorithm to bite the bullet and just put them in one at a time.

The algorithm described as “Biting the bullet” would be not sorting the new applicants and performing a insertion sort. Each new applicant would be compared to the current applicants until the appropriate position was discovered and then it woudl be inserted. This would repeat until the new applicants were inserted at the correct positions.

3) Estimate the complexity of each. Which one is better? What if the number (200/50) changes?

With the current number (200/50), sorting the new applicants then placing them in the already sorted stack seems better. Therefore, by sorting the 50 new applicants first, you would only have to go through the stack of 200 pre-sorted applicants once. Whereas, if you were to do the second method of sorting, you would have to go through the 200 pre-sorted application multiple times. However, if the number of unsorted applicants grew to something larger than 200, then you would have to basically go through a list much larger than the pre-sorted pile multiple times just to sort it. In this case, then it may be easier to just do the second method and go through the pre-sorted application multiple times, and the larger unsorted application pile once.

If we consider the current case where there are 200 applications in order with 50 new applications, we can observe the following complexity levels. In the event insertion sort is utilized (as described in answer #2 above), each application would be thumbed through until the appropriate location is determined for the first new application. This would then be repeated multiple times until all the applications are placed. This would equate to 200(n), or in our case 200(50). This is 10,000 manual operations. If binary sort was utilized, as described in #1 above,

the following operation would apply. (200/2) -> (100/2) -> (50/2) -> (25/2) -> (12/2) -> (6/2) -> (3/2) -> (appropriate slot located and application inserted). This would be 8 manual operations, or 8(n). Again, in our case this would translate to 8(50) which is only 400 manual operations. From this we can see that binary search only requires 4% of the manual operations which insertion methodology calls for. If the 200/50 relationship changes, we can expect the percent efficiency standard of binary sort relative insertion or linear sort to remain at O(log n) / O(n), where O is the total number of applications and n are the new applications to be input.

4) Compare the algorithms you develop to standard computer algorithms. Which computer algorithm is similar to your manual algorithm? Why? If you don't think it is similar to any computer algorithms, explain why not.

The algorithm described for question #1 using bubble sort assumes using a digital system, because doing bubble sort by hand would be very inefficient. While bubble sort is fine for ordering a set of 50, it would be much more inefficient with larger sets, with the average order being O(n^2). This would equate to 2,500 manual operations, with another 400 for the binary insertion (explained in question 3). If you had a stack of papers to organize, you would naturally choose a much faster way.

A manual sorting algorithm, such as going through the 50 new applications and putting them in by hand would rely more on intuition and guessing than easily programmed into a simple algorithm. The closest would be a binary sort/insert algorithm, which is a common computer searching algorithm. For example, if the sorting method was alphabetically by last name in a filing cabinet and the last name started with “B”, then you would intuitively jump to a spot roughly 1/26th of the way through the stack, see if the desired slot is before or after your guess, and then repeat that process. The main difference is that humans can make a intuitive judgments about the pile a pick the “middle” divider accordingly, rather than always halfway through.

Standard computer sorting algorithms benefit from years of optimization, since every bit of efficiency is very valuable for large data sets. The natural, but hard to quantify intuition that allows humans to take shortcuts can be carefully programmed. The language Python, which is known for dealing with data well, uses an algorithm called Timsort as its default. With an average complexity of O(n log n), this would take roughly 85 manual operations, as opposed to bubble sort’s 2,500, to sort a pile of 50. Timsort is a hybrid of merge sort and insertion sort, and works by finding already ordered subsets of data referred to as “runs”, then merging them together. It is adaptive, meaning it will adjust based on how the data is distributed.