Assignment 3 Writeup

Time Complexity

Assignment 3 demanded three different sorts i.e Bubble, Shell, and quick w/ a queue / stack. Each algorithm is effective, but some are more effective than others in certain situations due to their time complexity.

Bubble Sort

The bubble sort is better when working w/ arrays that a short but is bad when sorting long arrays. The Worst case scenario for a Bubble Sort is a reversed-ordered array, where it must swap or bubble each element the largest way possible to the end of the array. Each "round" of bubble would be used to its fullest extent.

The best case scenario would be when the array is already sorted and the Bubble Sort only makes comparisons & no swaps.

Worst-Case Time Complexity: $O(n^2)$. This is because we iterate through all n elements n times.

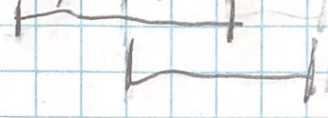
Best-Case Time Complexity: $O(n)$. This is because we only look through all n elements once.

Shell Sort

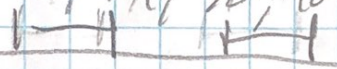
Shell Sort is generally better than bubble sort. The algorithm is unstable and the time complexity changes a lot depending on the input.

For **worst-case**: Consider array [32, 21, 70, 40]

In our first iteration we would use gap = 2 and do nothing [32, 21, 70, 40] => [32, 21, 70, 40]



Second Iteration we take gap = 1 and swap [32, 21, 70, 40] => [21, 32, 40, 70]



Worst case Time-Complexity: $O(n^2)$ because we had to iterate n elements n times. It essentially becomes a bubble sort.

Best case: The array is already sorted.

$O(n)$ is best case.

Quick Sort

On average the Quick Sort is better than the other sorts for large arrays. However it's worst case is $O(n^2)$.

Worst-Case Scenario: This occurs on a reverse ordered lists, when the partitions are skewed and when the min or max point is picked as a pivot.

For example, take arr = [2 4 3 1]
pivot

First iteration
=> [2,3,1] [4] [ ] => [2,3,1,4]
Left pivot Right Merge
pivot

Second
=> [2,1] [3] [4] => [2,1,3,4]
Left pivot Right Merge
pivot

Third
[1,2,3,4]

Worst-Case is $O(n^2)$ because we compared n elements n times

Best-Case: $O(n)$ where a list is already sorted.

Avg. Case: $O(n \log n)$ because we generally have our comparisons and pick decent pivots.

Graphs of Time Complexity

- **Bubble Sort**
  Shell Sort
  Quick Sort
  or ....

  Has ~~avg~~ best case $O(n)$
  Worst case $O(n^2)$

Shell Sort

Has avg. case = $O(n^{5/4})$
Best case $O(n)$
Worst case $O(n^2)$

- **Quick Sort**

  Has avg. case = $O(n \log n)$
  Best case = $O(n)$
  Worst case = $O(n^2)$

Analysis

Examing these graphs, it is important to note that number of steps is on the y-axis and the input size n of array is on the x-axis.

These graphs represent the growth rate, and thus we can clearly see why $O(n^2)$ (BLUE) is the worst case scenario because it requires the most steps almost all input sizes.

$O(n)$ (GREEN) is generally the best case when input size is greater than 10 because it requires less steps until then.

$O(\log(n))$ (PURPLE) is the most optimal and realistic case however which is accomplished in quicksort's avg. case. It is not only realistic but even faster than $O(n)$ when input size is less than 10.