**CSE 2046**

**Analysis of Algorithms**

**HW 2**

**(i) Sort all elements by Insertion-sort and return the 𝑘’th element in the list:**

We first apply insertion sort and then take the k’th element in the sorted list.

**Worst Case**: The worst-case of the insertion-sort occurs when the list in reverse order. In this case there is n(n-1)/2 comparisons and n(n-1)/2 swaps operations. Time complexity is **O(n2)**.

Example:

*5 4 3 2 1*

4 5 3 2 1

4 3 5 2 1

3 4 5 2 1

3 4 2 5 1

3 2 4 5 1

2 3 4 5 1

2 3 4 1 5

2 3 1 4 5

2 1 3 4 5

1 2 3 4 5

**Best Case:** The best-case of the insertion-sort occurs when the list already in order. In this case there is n-1 comparisons and no swap operations because it is already sorted. Time complexity is **O(n)**.

Example:

*1 2 3 4 5*

1 **2** 3 4 5

1 2 **3** 4 5

1 2 3 **4** 5

1 2 3 4 **5**

**Average Case:** Time complexity for average case is **O(n2)**

**Experiment:**

The experiment was made with the following data sizes: 8192, 16384, 32768, 65536, 131072, 262144 and 524288. Time has been measured as milliseconds. The results are as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Insertion Sort Search |  |  |  |  |  |  |  |
|  | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 |
| Best Case | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| Worst Case | 52 | 201 | 804 | 3313 | 12827 | 52936 | 209715 |
| Average Case | 7 | 35 | 126 | 468 | 1807 | 6789 | 28299 |

It can easily be seen that Worst-case and Average-case time complexities are O(n2) as expected in the theoretical part.

**(ii) Sort all elements by Merge-sort and return the 𝑘’th element in the list:**

We first apply merge sort and then take the k’th element in the sorted list.

**Worst Case**: The worst-case of the merge-sort occurs when merging two sets which are alternating. Even in bad case the worst case in merge sort is **O(nlogn)**.

Example:

5 1 7 3 6 2 8 4

5 1 7 3 6 2 8 4

5 1 7 3 6 2 8 4

1 5 3 7 2 6 4 8

1 3 5 7 2 4 6 8

1 2 3 4 5 6 7 8

For two elements sets we have 1 swap each total n/2 swaps, for 4 elements we have n/4 times 3 swaps etc. So

Total swap operations in this case is:

where n = 2k and that leads nlogn – 3n which is O(nlogn)

Number of comparisons are also O(nlogn).

**Best Case:** The best-case of the merge-sort occurs when the list already in order. In this case there is O(nlogn) comparisons and no swap operations because it is already sorted. Time complexity is **O(nlogn)**.

**Average case:** Time complexity for average case is **O(nlogn)**

**Experiment:**

The experiment was made with the following data sizes: 8192, 16384, 32768, 65536, 131072, 262144 and 524288. Time has been measured as milliseconds. The results are as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Merge Sort Search |  |  |  |  |  |  |  |
|  | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 |
| Best Case | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| Worst Case | 10 | 19 | 41 | 93 | 267 | 640 | 1515 |
| Average Case | 5 | 16 | 38 | 84 | 203 | 463 | 986 |

It can easily be seen that Worst-case and Average-case time complexities are close to O(nlogn) as expected in the theoretical part.

**(iii) Sort all elements by Quick-sort and return the 𝑘’th element in the list. While partitioning choosing the pivot element as the first element in an array.**

We first apply quick sort with pivoting as first element and then take the k’th element in the sorted list.

**Worst Case:** The worst-case of quick sort with first element pivoting occurs if the list is sorted in ascending or descending order. In such a case the time complexity is becoming **O(n2)**.

Example:

By selecting first element as pivot in each set:

1 2 3 4 5 6 7 8

**1** 2 3 4 5 6 7 8

1 **2** 3 4 5 6 7 8

1 2 **3** 4 5 6 7 8

1 2 3 **4** 5 6 7 8

1 2 3 4 **5** 6 7 8

1 2 3 4 5 **6** 7 8

1 2 3 4 5 6 **7** 8

So (n-1) + (n-2) + …. 1 comparisons make n(n-1)/2 comparisons which is O(n2).

**Best Case:** The best-case of quick sort with first element pivoting occurs when the first element in each set is the middle element of the set itself. In this case pivot is partitioning its set into two equal groups. In this case total comparisons are approximately n + 2 \* n/2 + 4 \* n/4 + … + 2k \* n/2k + … + n \* n/n, totally logn + 1 terms so nlogn + n which is **O(nlogn)** time complexity.

Example:

4 2 1 3 6 5 7 8

2 1 3 **4** 6 5 7 8

1 **2** 3 4 5 **6** 7 8

1 2 3 4 5 6 **7** 8

**Average case:** Time complexity for average case is **O(nlogn)**

**Experiment:**

The experiment was made with the following data sizes: 8192, 16384, 32768, 65536, 131072, 262144 and 524288. Time has been measured as milliseconds. The results are as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Quick Sort Search |  |  |  |  |  |  |  |
|  | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 |
| Best Case | 2 | 2 | 2 | 5 | 10 | 19 | 48 |
| Worst Case | 18 | 72 | 308 | 1138 | 4838 | 18883 | 73867 |
| Average Case | 2 | 4 | 6 | 16 | 30 | 42 | 90 |

Worst-case time complexity is O(n2) as expected in the theoretical part. Best-case and Average-case are close to O(nlogn) as seen in the table.

**(iv) Apply partial selection-sort, i.e. find the minimum element 𝑘 times to find the 𝑘’th smallest element.**

We first find the smallest element and then second smallest element and so on until we find kth smallest element.

**Worst Case:** Partial selection-sort always makes the same number of comparisons for any list. It does not depend if it is sorted or not. We find the least element in the set by comparing with all other elements in the set. For the first minimum, n-1, second n-2, … for the kth minimum n-k comparisons done.

Total comparisons: (n-1) + (n-2) + ... + (n-k) = kn – k(k+1)/2

And totally k swaps occur.

So the worst case is when kth element is the nth element. **O(n2)**

**Best Case:** Partial selection-sort always makes the same number of comparisons for any list. It does not depend if it is sorted or not. We find the least element in the set by comparing with all other elements in the set. For the first minimum, n-1, second n-2, … for the kth minimum n-k comparisons done.

Total comparisons: (n-1) + (n-2) + ... + (n-k) = kn – k(k+1)/2

And totally k swaps occur.

So the best case is when kth element is the 1st element. **O(n)**

**Average Case:** Partial selection-sort always makes the same number of comparisons for any list. It does not depend if it is sorted or not. We find the least element in the set by comparing with all other elements in the set. For the first minimum, n-1, second n-2, … for the kth minimum n-k comparisons done.

Total comparisons: (n-1) + (n-2) + ... + (n-k) = kn – k(k+1)/2

And totally k swaps occur.

So the average case is. **O(kn)**

**Experiment:**

The experiment was made with the following data sizes: 8192, 16384, 32768, 65536, 131072, 262144 and 524288. Time has been measured as milliseconds. The results are as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Partial Selection Sort Search | |  |  |  |  |  |  |
|  | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 |
| Best Case | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| Worst Case | 17 | 74 | 319 | 1139 | 4414 | 16723 | 62871 |
| Average Case | 17 | 82 | 260 | 804 | 2625 | 11425 | 46035 |

Worst-case time and average-case time complexities are O(n2) as expected in the theoretical part.

**(v) Apply partial heap-sort, i.e. store all elements in a max-heap and apply 𝑛−𝑘 times max removal. Return the max element in the root.**

First max-heapify the list then in each turn remove the maximum element until the kth minimum element.

**Worst Case:** Heap sort has the worst case complexity of O(nlogn). Partial heap-sort has a time complexity of O((n-k+1)logn + n). (+n comes from making firstly the list heap) When thinking about worst case, we have the worst case scenario when k is the first element in the list. When k = 1, we have a time complexity of **O(nlogn)**.

**Best Case:** Heap sort has the best case complexity of O(nlogn). Partial heap-sort has a time complexity of O((n-k+1)logn + n). (+n comes from making firstly the list heap) When thinking about best case, we have the best case scenario when k is the nth element in the list. When k = n, we have a time complexity of logn + n = **O(n)**.

**Average Case:** Heap sort has the best case complexity of O(nlogn). Partial heap-sort has a time complexity of O((n-k+1)logn + n). (+n comes from making firstly the list heap) When thinking about average case, we have the average case scenario when k is the n/2 th element in the list. When k = n/2, we have a time complexity of O(nlogn + n) = **O(nlogn)**.

**Experiment:**

The experiment was made with the following data sizes: 8192, 16384, 32768, 65536, 131072, 262144 and 524288. Time has been measured as milliseconds. The results are as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Partial Heap Sort Search | |  |  |  |  |  |  |
|  | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 |
| Best Case | 0 | 0 | 0 | 1 | 2 | 3 | 5 |
| Worst Case | 0 | 2 | 3 | 7 | 17 | 35 | 97 |
| Average Case | 0 | 1 | 3 | 5 | 11 | 32 | 81 |

Worst-case time and average-case time complexities are close to O(nlogn) as expected in the theoretical part.

**(vi) Not sort the list, but apply quick select algorithm, which is based on array partitioning, as described in the class. While partitioning, choose the pivot element as the first element in an array.**

**Worst Case:** The worst-case of quick select with first element pivoting occurs if the list is sorted ascending and we are searching the last element or the list is descending and we are searching for the last element.

Example:

Searching for last (8th) smallest element:

1 2 3 4 5 6 7 8

**1** 2 3 4 5 6 7 8

1 **2** 3 4 5 6 7 8

1 2 **3** 4 5 6 7 8

1 2 3 **4** 5 6 7 8

1 2 3 4 **5** 6 7 8

1 2 3 4 5 **6** 7 8

1 2 3 4 5 6 **7** 8

Its time complexity is **O(n2)**.

**Best Case:** The best-case of quick select with first element pivoting occurs if searched kth element is the pivot itself. Its time complexity is **O(n)**.

Example:

Searching 4th element in the list:

4 3 2 1 8 7 6 5

3 2 1 **4** 8 7 6 5

Finished. Totally n-1 comparisons. Complexity is O(n)

**Average Case:** The average case time complexity for the quick select is **O(n)**.

**Experiment:**

The experiment was made with the following data sizes: 8192, 16384, 32768, 65536, 131072, 262144 and 524288. Time has been measured as milliseconds. The results are as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Quick Select First Search | |  |  |  |  |  |  |
|  | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 |
| Best Case | 1 | 1 | 1 | 1 | 2 | 3 | 5 |
| Worst Case | 22 | 59 | 285 | 1096 | 4244 | 17539 | 72689 |
| Average Case | 0 | 1 | 1 | 2 | 4 | 9 | 19 |

Worst-case time complexity is O(n2) as expected in the theoretical part. Best and average case can be O(n) as seen in the table.

**(vii) Apply quick select algorithm, but this time use median-of-three pivot selection:**

**Worst Case:** The worst-case of quick select with median pivoting occurs if the first, middle and last term of the set are the smallest or the biggest elements in the set

Example:

Searching for last (8th) smallest element:

1 4 5 2 6 7 8 3

**1** 4 5 **2** 6 7 8 **3**

1 **2** 4 5 6 7 8 3

1 2 **4** 5 **6** 7 8 **3**

1 2 3 **4** 5 6 7 8

1 2 3 4 **5 6** 7 **8**

1 2 3 4 5 **6** 7 8

1 2 3 4 5 6 **7** **8**

1 2 3 4 5 6 7 8

Its time complexity is **O(n2)**.

**Best Case:** The best-case of quick select with median element pivoting occurs if searched kth element is the pivot itself in the median of first, last and middle elements in the list. Its time complexity is **O(n)**.

Example:

Searching 4th element in the list:

4 3 1 2 8 6 5 7

**4** 3 1 **2** 8 6 5 **7**

3 1 2 **4** 8 6 5 7

Finished. Totally n-1 comparisons. Complexity is O(n)

**Average Case:** The average case time complexity for the quick select is **O(n)**.

**Experiment:**

The experiment was made with the following data sizes: 8192, 16384, 32768, 65536, 131072, 262144 and 524288. Time has been measured as milliseconds. The results are as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Quick Select Median Search | |  |  |  |  |  |  |
|  | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 |
| Best Case | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| Worst Case | 20 | 54 | 263 | 989 | 3817 | 15643 | 65702 |
| Average Case | 0 | 1 | 1 | 1 | 1 | 2 | 5 |

Worst-case time complexity is O(n2) as expected in the theoretical part. Best and average case can be O(n) as seen in the table.