# Question 3 – Checklist Table

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Tasks** | **Mark** | **Checklist**  **(Yes/no) and person(s) to work on this task** |
| 1 | Quick-select algorithm for searching the *k*-th element.  - Correct implementation of searching *k*-th element using quick-select algorithm.  - Output the intermediate results for a random array of 10 elements for tutor to inspect the correctness of the algorithms. | 4 | Yes  (Fang Jee, Michelle) |
| 2 | Test the Quick-select algorithm with  - Different array sizes (10,000, 100,000, 1,000,000, etc.) that can show significant results.  - Different pivot (random pivot vs fixed pivot).  - Different cases (e.g. best, average, and worst). | 4 | Yes  (Fang Jee, Michelle) |
| 3 | Merge-sort algorithm for searching the *k*-th element.  - Correct implementation of searching *k*-th element using merge-sort algorithm.  - Output the intermediate results for a random array of 10 elements for tutor to inspect the correctness of algorithms. | 4 | Yes  (Justin, Kah Ming) |
| 4 | Test the Merge-sort algorithm with  - Different array sizes (10,000, 100,000, 1,000,000, etc.) that can show significant results.  - Test the Merge-sort algorithm with different cases (e.g. best, average, and worst). | 4 | Yes  (Justin, Kah Ming) |
| 5 | - Include the above experiment results that can be used to perform a comparative analysis (such as drawing the graphs for comparison) between the two algorithms (Quick-select & Merge-sort) in the report.  - Conclude your findings in the report. | 4 | Yes  (Fang Jee , Justin, Kah Ming, Michelle) |

# Quick-select algorithm

## Array size test (k = 80)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Array size, *n*** | **10,000** | **100,000** | **1,000,000** | **5,000,000** |
| Attempt 1 (s) | 0.000999 | 0.001983 | 0.033009 | 0.170038 |
| Attempt 2 (s) | 0.000992 | 0.002001 | 0.033166 | 0.179039 |
| Attempt 3 (s) | 0.001001 | 0.002001 | 0.034009 | 0.182043 |
| Attempt 4 (s) | 0.000984 | 0.003 | 0.036008 | 0.183031 |
| Attempt 5 (s) | 0.001001 | 0.002 | 0.035008 | 0.17803 |
| Average Duration (s) | 0.0009954 | 0.002197 | 0.03424 | 0.1784362 |

## Pivot test (same set of array size = 10000)

|  |  |  |
| --- | --- | --- |
| **Pivot selection** | **Random** | **Fixed** |
| Attempt 1 (s) | 0.001001 | 0.113026 |
| Attempt 2 (s) | 0.001 | 0.112027 |
| Attempt 3 (s) | 0.001 | 0.112024 |
| Attempt 4 (s) | 0.001 | 0.114027 |
| Attempt 5 (s) | 0.001 | 0.112027 |
| Average Duration (s) | 0.001 | 0.112626 |

## Test of different cases (same set of array size = 10000)

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Best** | **Average** | **Worst** |
| Attempt 1 (s) | 0 | 0  (k=37) | 0.112036 |
| Attempt 2 (s) | 0 | 0.001 (k=1238) | 0.112026 |
| Attempt 3 (s) | 0.001001 | 0.001001 (k=4632) | 0.112036 |
| Attempt 4 (s) | 0 | 0  (k=9574) | 0.112024 |
| Attempt 5 (s) | 0.000998 | 0.001 (k=58) | 0.111036 |
| Average duration (s) | 0.0003998 | 0.0006002 | 0.1118316 |

# Merge-sort algorithm

## Array size test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Array size, *n*** | **10,000** | **100,000** | **1,000,000** | **5,000,000** |
| Attempt 1 (s) | 0.007011 | 0.077005 | 55.4115 | 1191.9 |
| Attempt 2 (s) | 0.007002 | 0.071027 | 54.8905 | 1197.78 |
| Attempt 3 (s) | 0.007002 | 0.072017 | 56.7984 | 1179.17 |
| Attempt 4 (s) | 0.006010 | 0.071016 | 56.687 | 1183.14 |
| Attempt 5 (s) | 0.007001 | 0.072005 | 56.7014 | 1177.73 |
| Average Duration (s) | 0.006805 | 0.072614 | 56.09776 | 1185.944 |

## Test of different cases (same set of array size = 10000)

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Best** | **Average** | **Worst** |
| Attempt 1 (s) |  |  |  |
| Attempt 2 (s) |  |  |  |
| Attempt 3 (s) |  |  |  |
| Attempt 4 (s) |  |  |  |
| Attempt 5 (s) |  |  |  |
| Average duration (s) |  |  |  |

# Comparative analysis

Figure 1: Running time test for different array sizes

According to the graph above, the merge-sort algorithm’s running time increases exponentially after the array size of 1mil, where it takes around 56s to complete, and around 20mins for the array size of 5mil. On the other hand, quick-select algorithm runs quickly, less than 1s for all the different array sizes. The result reflects that time complexity of quick-select algorithm is less than merge-sort algorithm; time complexity of quick-select algorithm is O(*n*), while time complexity of merge-sort algorithm is O(*n lg* *n*).

Figure 2: Different pivot selection for quick-select algorithm

For quick-select algorithm, fixed pivot runs at a consistent speed of 0.001s, while random pivot runs at less than 0.001s when the k = 5,690 and 7,653. Using a random pivot performs 40.17% faster as compared to using a fixed pivot, which is the last element of the partition in this case. In conclusion, random pivot runs slightly faster than fixed pivot when the array size is 10,000.

Figure 3: Different cases analysis

For quick-select algorithm, it performs at a similar speed for best and average case, where the time complexity is given by O(*n*). For the worst case, quick-select algorithm performs significantly slower as compared to the other two cases, where the time complexity is given by O(*n*2); the time difference between best case and worst case is 0.1114318s, which is roughly 278 times of the best case’ time of 0.0003998s.

# Conclusion

In terms of improvement for our algorithms, for the partition code of quick-select algorithm, the partition through using three sequences (L, E, G) could be used instead to achieve a better representation of the sorted array.

In conclusion, quick-select algorithm and merge-sort algorithm perform at a similar speed when dealing with a small size of input. Quick-select algorithm is more efficient in finding a specific (kth smallest element) as compared to merge-sort algorithm since not all the elements must be sorted. However, quick-select algorithm performs worse than merge-sort algorithm when it comes to worst case; time complexity for quick-select algorithm’s worst case is O(*n*2), while time complexity of merge-sort algorithm is O(*n lg* *n*).