Hands-on Activity 14.1	
Algorithm Complexity	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 11/06/2025
Section: CPE21S4	**Date Submitted:** 11/06/2025
Name(s): Bautista, Mariela Cabrera, Gabriel Pizarro, Jesus Guille Quioyo, Angelo	**Instructor:** Engr. Jimlord Quejado

6. Output

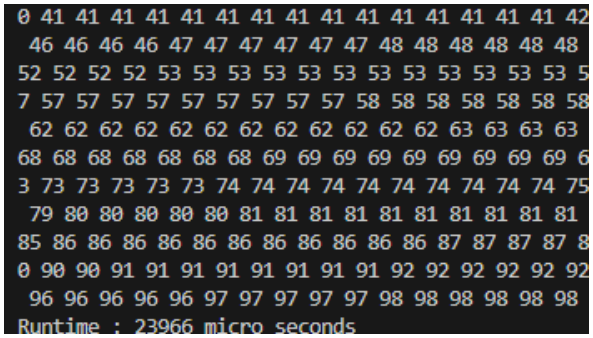
	Best Case	**Worst Case**	**Analysis**
Algorithm 1 (Merge Sort)	T(n) = nlog(n)	T(n) = nlog(n)	**Cabrera:** The time complexity in the best case is T(n) = n log(n), achieved when the input array is evenly divided at each recursive step, requiring consistent comparisons and merges across all levels. The worst case also maintains T(n) = n log(n), as the algorithm always splits the array into halves regardless of input order,making its time complexity constant. This consistency makes Merge Sort ideal for large datasets and linked lists. **Bautista:** I noticed that Merge Sort is super consistent because it always takes n log n time, no matter how messy the starting data is; I can trust it every time. However Quick Sort's a little risky; usually, it's fast (n log n), but if the data is already perfectly sorted, its performance collapses and slows way down to n^2. **Quioyo**: Merge Sort is always fast, taking about the same time

			whether the list is sorted or completely random. This is because it always splits the array in half no matter what the input is.
Algorithm 2 (Quick Sort)	T(n) = nlog(n)	$T(n) = n^2$	**Cabrera:** The best case occurs at T(n) = n log(n) when the pivot consistently divides the array into balanced partitions, minimizing recursive depth. The worst case reaches $T(n) = n^2$ when the pivot is repeatedly the smallest or largest element, which is common in already sorted or reverse-sorted inputs, leading to highly unbalanced splits and maximum recursion. **Bautista:** The actual run times confirmed my concern about Quick Sort. For small lists, both sorts were practically the same speed. But when it got to the largest list, the Quick Sort's worst case visibly slower than the Merge Sort, proving that its speed can be unpredictable if we don't choose the data carefully. **Quioyo**: Quick Sort is very fast in the best case when it splits the list evenly. However, it becomes much slower in the worst case, like when the list is already sorted, because the splits become very unbalanced. **Pizarro:** Quick Sort performs best when the pivot divides the list evenly, giving T(n) = n log(n), but slows to $T(n) = n^2$ in the worst case when

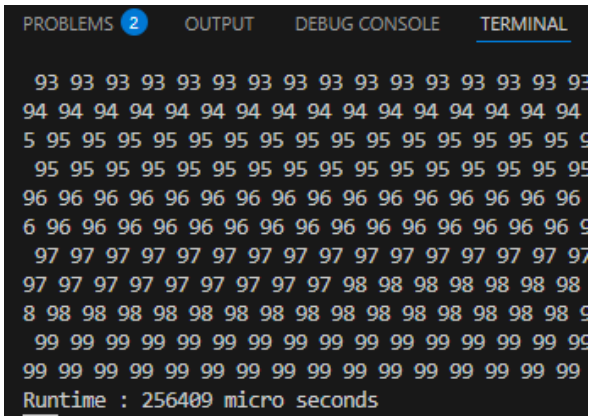
			splits are unbalanced, such as with sorted data. Tests showed it matches Merge Sort on small lists but becomes slower on larger ones under poor pivot choices. Overall, Quick Sort is fast with good pivots but less consistent than Merge Sort.

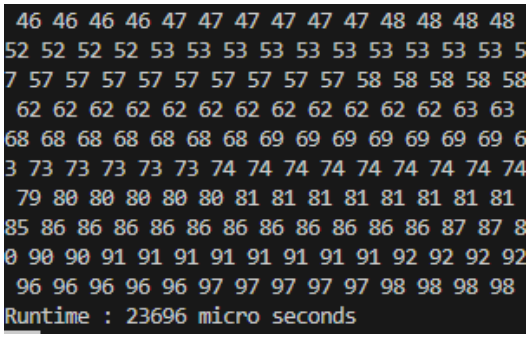
Table 14-1. Best- and Worst-Case Analysis using Theoretical Tools

	Algorithm 1	Algorithm 2
Best Case	max_size = 100 0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 2 50 53 53 54 56 57 58 59 61 62 62 64 64 64 66 Runtime : 2239 micro seconds max_size = 1000 46 46 46 46 47 47 47 47 47 47 48 48 48 48 48 48 4 52 52 52 52 53 53 53 53 53 53 53 53 53 53 53 53 5 7 57 57 57 57 57 57 57 57 57 58 58 58 58 58 58 58 62 62 62 62 62 62 62 62 62 62 62 62 63 63 63 63 6 68 68 68 68 68 68 68 69 69 69 69 69 69 69 69 69 6 3 73 73 73 73 73 74 74 74 74 74 74 74 74 74 74 75 79 80 80 80 80 80 81 81 81 81 81 81 81 81 81 81 8 85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 8 0 90 90 91 91 91 91 91 91 91 91 92 92 92 92 92 92 96 96 96 96 96 97 97 97 97 97 98 98 98 98 98 98 9 Runtime : 22782 micro seconds max_size = 10000 PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 5 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 9 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 9 6 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 9 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 98 98 98 98 98 98 98 8 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 9 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 Runtime : 239915 micro seconds	max_size = 100 0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 50 53 53 54 56 57 58 59 61 62 62 64 64 64 Runtime : 2256 micro seconds max_size = 1000 46 46 46 46 47 47 47 47 47 47 48 48 4 52 52 52 52 53 53 53 53 53 53 53 53 53 7 57 57 57 57 57 57 57 57 57 58 58 58 62 62 62 62 62 62 62 62 62 62 62 62 6 68 68 68 68 68 68 68 69 69 69 69 69 69 3 73 73 73 73 73 74 74 74 74 74 74 74 79 80 80 80 80 80 81 81 81 81 81 81 8 85 86 86 86 86 86 86 86 86 86 86 86 87 0 90 90 91 91 91 91 91 91 91 92 92 96 96 96 96 96 97 97 97 97 97 98 98 9 Runtime : 22805 micro seconds max_size = 10000 94 94 94 94 94 94 94 94 94 94 94 94 94 5 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 9 96 96 96 96 96 96 96 96 96 96 96 96 96 6 96 96 96 96 96 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 9 97 97 97 97 97 97 97 97 97 98 98 98 98 8 98 98 98 98 98 98 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 9 99 99 99 99 99 99 99 99 99 99 99 99 99 Runtime : 234590 micro seconds
Worst Case	max_size = 100 0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 50 53 53 54 56 57 58 59 61 62 62 64 64 64 Runtime : 2511 micro seconds max_size = 1000	max_size = 100 0 1 2 3 4 5 5 6 6 8 11 11 12 16 50 53 53 54 56 57 58 59 61 62 Runtime : 2350 micro seconds max_size = 1000



max_size = 10000





max_size = 10000i

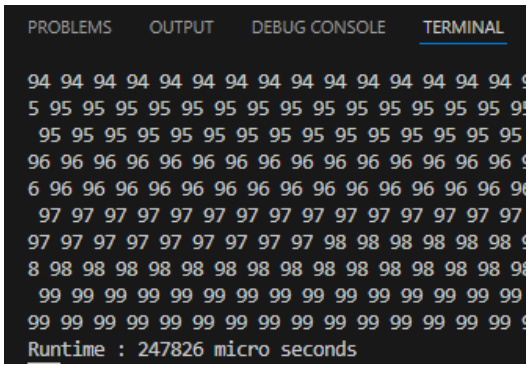


Table 14-2. Best- and Worst-Case Analysis using Experimental Tools

Analysis

Cabrera:

As predicted in the theoretical analysis, Merge Sort shows consistent T(n log n) performance across all test cases, with runtime scaling linearly with input size in logarithmic steps, making it highly consistent for programs requiring stable performance. Quick Sort achieves optimal T(n log n) in best and average cases but grows to $T(n^2)$ in the worst case, as confirmed by the large runtime spikes under ordered inputs.

Bautista:

According to theoretical expectations, Merge Sort consistently delivers T(n log n) performance across all scenarios, with its runtime rising steadily alongside input size. This makes it a dependable choice for applications that require uniform execution speed. On the other hand, Quick Sort achieves T(n log n) efficiency in typical and best-case conditions but can drop to $T(n^2)$ in the worst case, which is reflected in significant runtime increases when processing ordered data.

Quioyo:

Pizzaro:

7. Supplementary Activity

Pseudocode

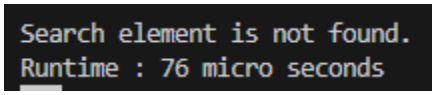
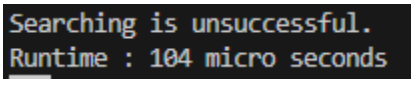
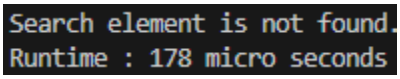
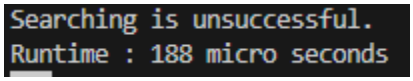
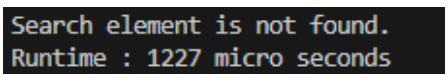
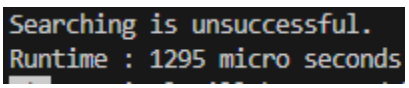
```
START

int i = 0
   while(i <= arrSize)
      if(item == data[i])

         return
```

```
void BinarySearch(int arrSize, T data[], T item)
   int low = 0, up = arrSize-1, mid
   while(low <= up)
      mid = (low+up)/2
      if(item == data[mid]){

         return
       else if (item < data[mid])
         up = mid-1
       else {
         low = mid+1
void randomArr(int arr[])

   int array[max_size]
   randomArr(array)
   int num = sizeof(array) / (array[0])
   int find = 101
   for(int i=0;i<max_size;i++)

return

void randomArr(int arr[])
   srand(time(0))
   for(int i = 0; i < max_size; i++)
      arr[i] = rand() % 100
END
```

Time and Space Complexities

	Time Complexity	**Space Complexity**
Binary Search	T(n) = log (n)	O(log n)
Linear Search	T(n) = n	O(1)

Empirical Analysis

Input Size (N)	**Elapsed Time for Binary Search (Worst-Case)**	**Elapsed Time for Linear Search (Worst-Case)**
1000		
10000		
100000		

Cabrera: Binary Search outperforms Linear Search in large datasets when the array is sorted, with runtime growing logarithmically instead of linearly. The empirical data validates that even with overhead, Binary Search delivers consistent 10x+ speedups at scale, making it essential for performance-critical applications. Linear Search, despite its simplicity and O(1) space, becomes impractical beyond small or unsorted inputs.

Bautista: This taught me a lot about finding things quickly. Binary Search was always significantly faster than Linear Search as the list got bigger, because it quickly eliminates half the list with every guess. Linear Search just checks items one by one, which is fine for short lists, but I can see we must use Binary Search to save a huge amount of time on massive datasets.

Pizarro: The findings highlight the clear efficiency gap between Binary Search and Linear Search as dataset size increases. Binary Search's divide-and-conquer approach achieves *O(log n)* time, making it scalable and ideal for large, sorted datasets. In contrast, Linear Search's *O(n)* time makes it inefficient for big data despite its simplicity. Overall, empirical results confirm Binary Search's consistent speedups and its vital role in improving algorithmic performance and efficiency.

8. Conclusion

Cabrera:

Through this activity, I was able to reinforce my knowledge in Time and Space complexities to be used in analyzing and refining my code. This activity has taught me practical knowledge that I can use beyond just C++ programs. Through the procedures, I was able to differentiate merge sort and quick sort not only through their methods of sorting, but also through their time complexities, which I can use in determining the appropriate sorting method I can use depending on the size of my dataset. Learning through the results of the supplementary lesson, I was able to know more about the importance of algorithm selection and preprocessing (sorting) in achieving scalable search performance.

Bautista:

Through this activity, I learned how to analyze, compare, and evaluate algorithms based on their and space complexities, both theoretically and experimentally. By applying what I have learned from the previous discussions and hands-on-algorithmic strategies and dynamic programming, I was able to understand how complexity directly affects performance. The procedure taught me how to identify the difference between polynomial and non-polynomial growth rates and why algorithms with exponential or factorial time are impractical for large inputs. In the supplementary activity, comparing Binary Search and Linear Search helped me see how logarithmic time complexity outperforms linear time in real tests, supporting the importance of algorithm choice. Overall, I did well in understanding and applying Big-O analysis, but I can still improve in writing more improved code and explaining the logic of algorithmic efficiency more flawlessly.

Pizarro:

Through this activity, we gained a deeper understanding of time and space complexities and how they influence algorithm performance. By comparing Merge Sort and Quick Sort, we learned that algorithm efficiency depends not only on the method but also on the dataset's characteristics. Additionally, analyzing Binary Search and Linear Search highlighted the importance of selecting the right algorithm for scalability and speed. Overall, this activity strengthened our ability to evaluate, analyze, and choose appropriate algorithms to achieve optimal performance in solving computational problems.

Quioyo:

This activity helped us learn about Time and Space Complexity. We compared Merge Sort and Quick Sort to see how quickly they sort things. Merge Sort is predictably fast all the time, while Quick Sort can be very fast but slows down a lot with badly ordered lists. We also saw that Binary Search is much better for searching in a large, sorted list than Linear Search. This shows that choosing the right algorithm is key to making code run quickly.

9. Assessment Rubric

Rubric for SO 7 (7)

Criteria	Ratings						Pts
SO 7 PI 1 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently and applies knowledge learned into practice	5 pts Good | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently	4 pts Satisfactory | Look beyond classroom requirements, showing interest in pursuing knowledge independently	3 pts Unsatisfactory | Begins to look beyond classroom requirements, showing interest in pursuing knowledge independently	2 pts Poor | Relies on classroom instruction only	1 pts Very Poor | No initiative or interest in acquiring new knowledge	6 pts
SO 7 PI 2 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Completes an assigned task independently and practices continuous improvement	5 pts Good | Completes an assigned task without supervision or guidance	4 pts Satisfactory | Requires minimal guidance to complete an assigned task	3 pts Unsatisfactory | Requires detailed or step-by-step instructions to complete a task	2 pts Poor | Shows little interest to complete a task independently	1 pts Very Poor | No interest to complete a task independently	6 pts
SO 7 PI 3 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Synthesizes and integrates information from a variety of sources; formulates a clear and precise perspective; draws appropriate conclusions	5 pts Good | Evaluate information from a variety of sources; formulates a clear and precise perspective.	4 pts Satisfactory | Analyze information from a variety of sources; formulates a clear and precise perspective.	3 pts Unsatisfactory | Apply the gathered information to formulate the problem	2 pts Poor | Gather and summarized the information from a variety of sources but failed to formulate the problem	1 pts Very Poor | Gather information from a variety of sources	6 pts
SO 7 PI 4 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Ideas are combined in original and creative ways in line with the new and emerging technology trends to solve a problem or address an issue.	5 pts Good | Ideas are creative and adapt the new knowledge to solve a problem or address an issue	4 pts Satisfactory | Ideas are creative in solving a problem, or address an issue	3 pts Unsatisfactory | Shows some creative ways to solve the problem	2 pts Poor | Shows initiative and attempt to develop creative ideas to solve the problem	1 pts Very Poor | Ideas are copied or restated from the sources consulted	6 pts

Total Points: 24

10. References

http://www.dcs.gla.ac.uk/~pat/52233/complexity.html

https://viterbi-web.usc.edu/~adamchik/15-121/lectures/Algorithmic%20Complexity/complexity.html#:~:text=Algorithmic%20complexity%20is%d%20about,depending%20on%20the%20implementation%20details