1. **Theoretical question**
2. Insertion Sort

Best Case

For insertion sort, the best case will be the input is already in order, therefore, for each element inserted, only the previous element needs to be examined, so in the best case, the time complexity of insertion sort is

Worst Case

The worst case will be the input is totally reversed. In that case, the first element being considered when the second element is inserted and the first two elements being considered when the third element is inserted…Then the time complexity on the worst case will be

Average Case

For the general case, assuming that the input array is half reversed, the time complexity is the same as the worst case, which is

1. Selection Sort

Because selection sort divides the array into two parts, one is ordered and the other is unordered, the algorithm selects elements from the unordered array and inserts them into the ordered array, so its time complexity is

Due to there is two nested for loops in the selection sort, those three cases for selection sort will be the same.

1. Bubble Sort

For Bubble Sort, because there is two nested for loops, the time complexity will be the same as Selection Sort which is

1. Merge sort

For merge sort, the time complexity will be ) , where is the algorithm divides the array into two parts, and is the time complexity for the algorithm to conquer the divided parts. It will not be affected by the order of the input array. Therefore, the time complexity for those three cases will be the same.

1. Quick sort

Because quick sort is a recursion algorithm, therefore, the time performance of quicksort depends on the depth of the quicksort recursion.

Best Case

On the best case, Partition evenly divides the array into two parts every time. Therefore, if the size of input is n, then the time complexity will be

Average Case

On the average case, partition divides the array from somewhere in the middle, therefore, the time complexity will be the same as on the best case.

Worst Case

On the worst case, the input is sorted or reversed, one of the divided parts is empty. On this case the time complexity will be

However, we can make some changes on the partition function to avoid it happens.

1. Heapsort

Heap sort is the continuous construction of the heap, after finish constructed the first heap, swapping head and tail, then adjust the heap to make it to be a max heap or min heap, then removing the head(tail) and reconstructing the heap with the rest part.

The time complexity is , is the time complexity for the heap construction. is the time complexity for adjusting the heap. No matter how the order of the input is, the time complexity will stay the same.

1. Counting Sort

The time complexity for counting sort is where the is the range of the input. For counting sort, no matter how the order of the input is, the time complexity will stay the same.

1. Radix Sort

The time complexity for counting sort is ) where the is the number of passes and is the radix used (In this project, it is 10). For radix sort, no matter how the order of the input is, the time complexity will stay the same.

1. **Data generation and experimental setup**

* What kind of machine did you use?

I ran the code on the win 11 system, and I got Intel i7-9750H as my CPU which is 2.60GHz and the Ram on the machine is 32 GB. The IDE I had used was Microsoft Visual Studio 2019, and the libraries I used were:

#include<iostream>

#include<vector>

#include<string>

#include<climits>

#include<math.h>

#include<time.h>

#include<random>

* What timing mechanism?

The timing mechanism I used was the clock () method in the <time.h> library.

I added clock () before the test and added one more clock () after the experiment, then the after one minus that one before will be the execution time of the experiment.

* How many times did you repeat each experiment?

I repeated 5 times for each experiment, because the more repetition, the more precision.

* What times are reported?

The average times will be reported in the experiment.

* How did you select the inputs?

I chose 30000 as my small size, 60000 as my median size, 90000 as my large size. Because when I test different input, I found those algorithms will not have obvious differences for size under 30000. So, I putted 30000 as my small size, and median size will be two times of the small size, large size will be three times of the small size. For those sort algorithms that can not handle large size input very well, I chose 10000 as the small size, 20000 as the median size and 30000 as the large size.

* Did you use the same inputs for all sorting algorithms?

No, through analysis, the best five of the eight sorting algorithms are Merge sort, Quick sort (best version), Heap Sort, Counting Sort, and Radix Sort. So, I use smaller inputs for algorithms other than these five, because large outputs will make them run very slowly. For those five algoritm

I have three functions to produce different size and order of input.

Text

Description automatically generated

In\_order () produces the array in range from 0 to size - 1 and the elements is sorted. Reversed () produces the array in range from 0 to size – 1 and the element is reversed. Random () produces the array in range from 0 to size – 1 and the element is in the random order.

Insertion Sort:

Best Case:

Chart, line chart

Description automatically generated

Average Case:

Chart, line chart

Description automatically generated

Worst Case:

Chart, line chart

Description automatically generated

Selection:

Best Case:

Chart, line chart

Description automatically generated

Average Case:

Chart, line chart

Description automatically generated

Worst Case:

Chart, line chart

Description automatically generated

Bubble Sort:

Best Case:

Chart, line chart

Description automatically generated

Average:

Chart, line chart

Description automatically generated

Worst

Chart, line chart

Description automatically generated

Merge Sort:

Best:

Chart, line chart

Description automatically generated

Average:

Chart, line chart, scatter chart

Description automatically generated

Worst:

Chart, line chart

Description automatically generated

Quick Sort:

Best:

Chart, line chart

Description automatically generated

Average:

Chart, line chart

Description automatically generated

Worst

Chart, line chart

Description automatically generated

Heap Sort

Best

Chart, line chart

Description automatically generated

Average

Chart, line chart

Description automatically generated

Worst

Chart, line chart

Description automatically generated

Counting Sort

Best Case:

Chart, scatter chart

Description automatically generated

Average Case:

Chart, line chart

Description automatically generated

Worst:

Chart, line chart, scatter chart

Description automatically generated

Radix Sort

Best:

Chart, line chart

Description automatically generated

Average:

Chart, line chart, scatter chart

Description automatically generated

Worst:

Chart, line chart

Description automatically generated

Best Five Sorting Algorithm:

Best Case:

Average Case:

Worst Case: