Revised: 1/3/2019

Recursive Sorting

Michael C. Hackett

Computer Science Department

Community College of Philadelphia

Lecture Topics

- Recursive Bubble Sort
- Merge Sort
- Quicksort

Using Recursion to perform a Bubble Sort

- You’ve seen the bubble sort (and its variants) implemented using an iterative algorithm.

- Since any problem that can be solved iteratively can be solved recursively, we can design a recursive replacement for the iterative bubble sort.

Bubble Sort (Iterative Algorithm)

```
void bubbleSort(int a[], int length) {
    for(int i = 0; i < length; i++) {
        for(int j = 1; j < length; j++) {
            if(a[j-1] > a[j]) {
                int temp = a[j-1];
                a[j-1] = a[j];
                a[j] = temp;
            }
        }
    }
}
```

Using Recursion to perform a Bubble Sort

- For an ascending sort, the first pass will move the largest value to the end of the array (length - 1).
  - The next pass will move the second largest value to index length - 2.
  - The next pass will move the third largest value to index length - 3.
  - And so on…

- The last pass of the iterative algorithm will ensure the smallest value is placed in index 0.
  - At this point, the smallest value is guaranteed to already be in index 0.

Using Recursion to perform a Bubble Sort

- The base case is
  - When the algorithm sorts for index 0.
  - Since all other values will have already been sorted in the array, it implies the value at index 0 is already in the correct position.

- The recursive case is for all sorting other indexes.

Bubble Sort (Recursive Algorithm)

```
void bubbleSort(int a[], int length) {
    if(length == 1) {
        return;
    }

    for(int i = 0; i < length-1; i++) {
        if(a[i] > a[i+1]) {
            int temp = a[i+1];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }

    bubbleSort(a, length-1);
}
```

Using Recursion to perform a Bubble Sort

- This recursive bubble sort algorithm, like the iterative version, also performs in polynomial time.

- There are no advantages (speed/complexity) by using an a recursive bubble sort over an iterative bubble sort.
    - This is example is here mainly to show you the two different implementations (recursive vs iterative).

Recursive Sorting Algorithms

- We’ll now see a pair of comparative sorting algorithms, Merge Sort and Quicksort, that are typically implemented using recursion.
  - They can be done iteratively, but the code is much easier to read when implemented recursively.

- Both algorithms take a “divide and conquer” approach to sorting, much like the how the binary search algorithm performs its searches.

Merge Sort Algorithm

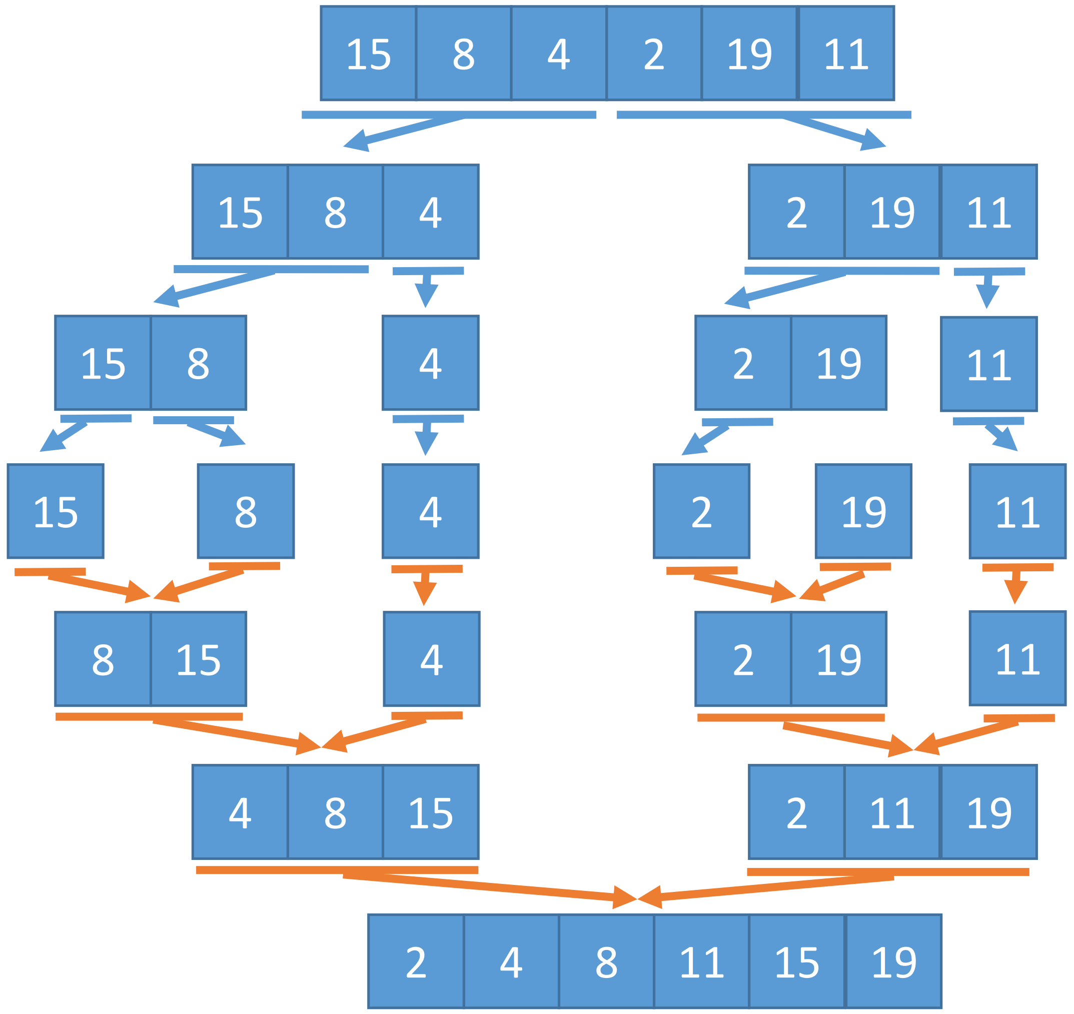
- In the Merge Sort algorithm, the array is repeatedly (recursively) split in half until it reaches halves that only contain one element.
  - At this point, the lowest depth has been reached.

- Then, working backwards, it sorts/merges the smaller arrays back together

Merge Sort Algorithm

- The image on the right gives the basic idea of how it works.
  - It divides up the array (Blue lines)
  - Then merges the array back together (Orange lines)

- Each merge involves two, sorted arrays.
  - Since the arrays to merge are in order, merging them is not computationally difficult.



Merge Sort Algorithm

- The C++ functions are a bit too long to put here in their entirety.
  - See the Sample Code provided.

- The next slides explain, *very briefly*, what is needed.
  - Two functions:
    - One function for the algorithm
    - Another function that handles the merging process

Merge Sort

```
mergesort(array, left, right):
    If left boundary < right boundary :
        Find the middle, m
        mergesort(array, left, m)
        mergesort(array, m+1, right)

        merge(array, left, m, right)
```

Merge Sort

```
merge(array, left, middle, right) :
        leftArray[middle - left + 1]
        rightArray[right - middle]
        Copy left side of array to leftArray
        Copy right side of array to rightArray
        Put the values of leftArray and rightArray back into array, in the correct order
        Put the remaining value, leftover in either leftArray or rightArray, into array
```

Quicksort Algorithm

- In the Quicksort algorithm, the array is repeatedly (recursively) split into two smaller partitions, until the partitions only contain one element.
  - At this point, the lowest depth has been reached.
- The algorithm chooses a value in each partition, called the **pivot**.
  - One of the two partitions will contain any values less than the pivot.
  - The other partition will contain any values greater than the pivot.
- The process repeats recursively until partitions of length 1 are reached.
  - At which point, the array will have been sorted through the pivot processes.

Quicksort Algorithm

- There are a few ways of selecting the pivot:
  - Always use the last element.
  - Always use the middle element.
  - Always use the first element.
  - Always use a randomly chosen element.
- The Sample Code provided uses the middle element as the selected pivot value.
- The next slides explain, *very briefly*, what is needed.

Quicksort

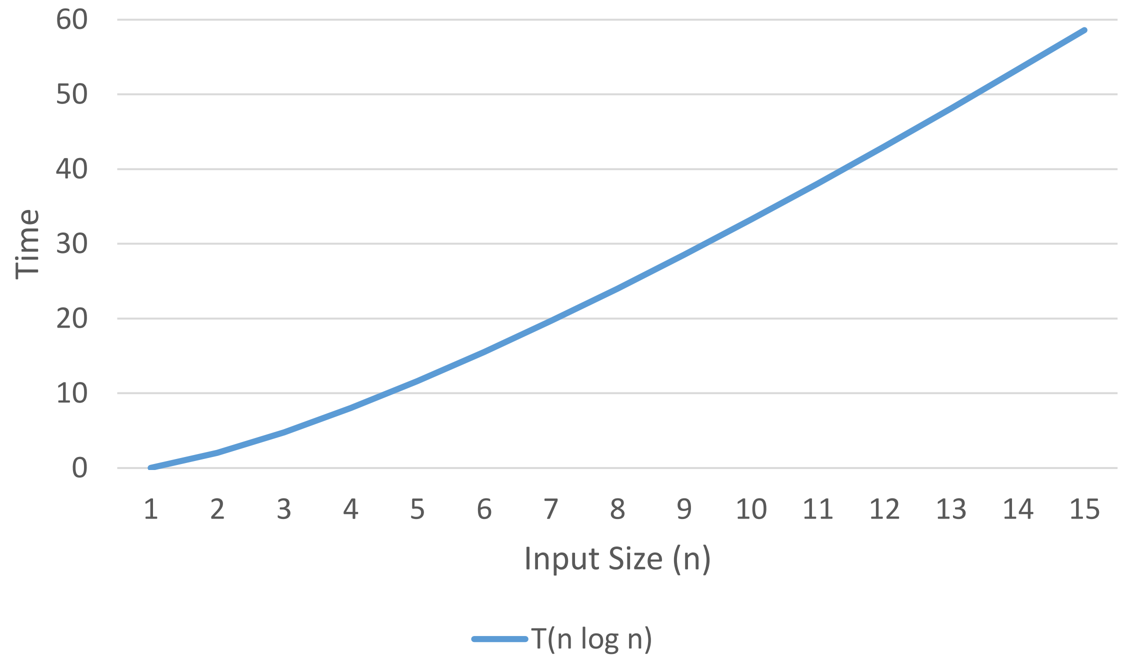
```
quicksort(array, start, end) :
        Select the pivot (the value in the middle)
        Partition the array
        quicksort(lower half)
        quicksort(upper half)
```

Merge Sort and Quicksort

- Both algorithms use the divide and conquer process like a binary search.
  - Which we already determined performs in logarithmic time.
- Both algorithms, at one point or another, will have partitions with a length of 1.
  - The algorithms will perform the logarithmic divide and conquer operations for as many elements that exist in the array.
- Their time complexity is a mix of logarithmic and linear time.

Quasi-Linear Time

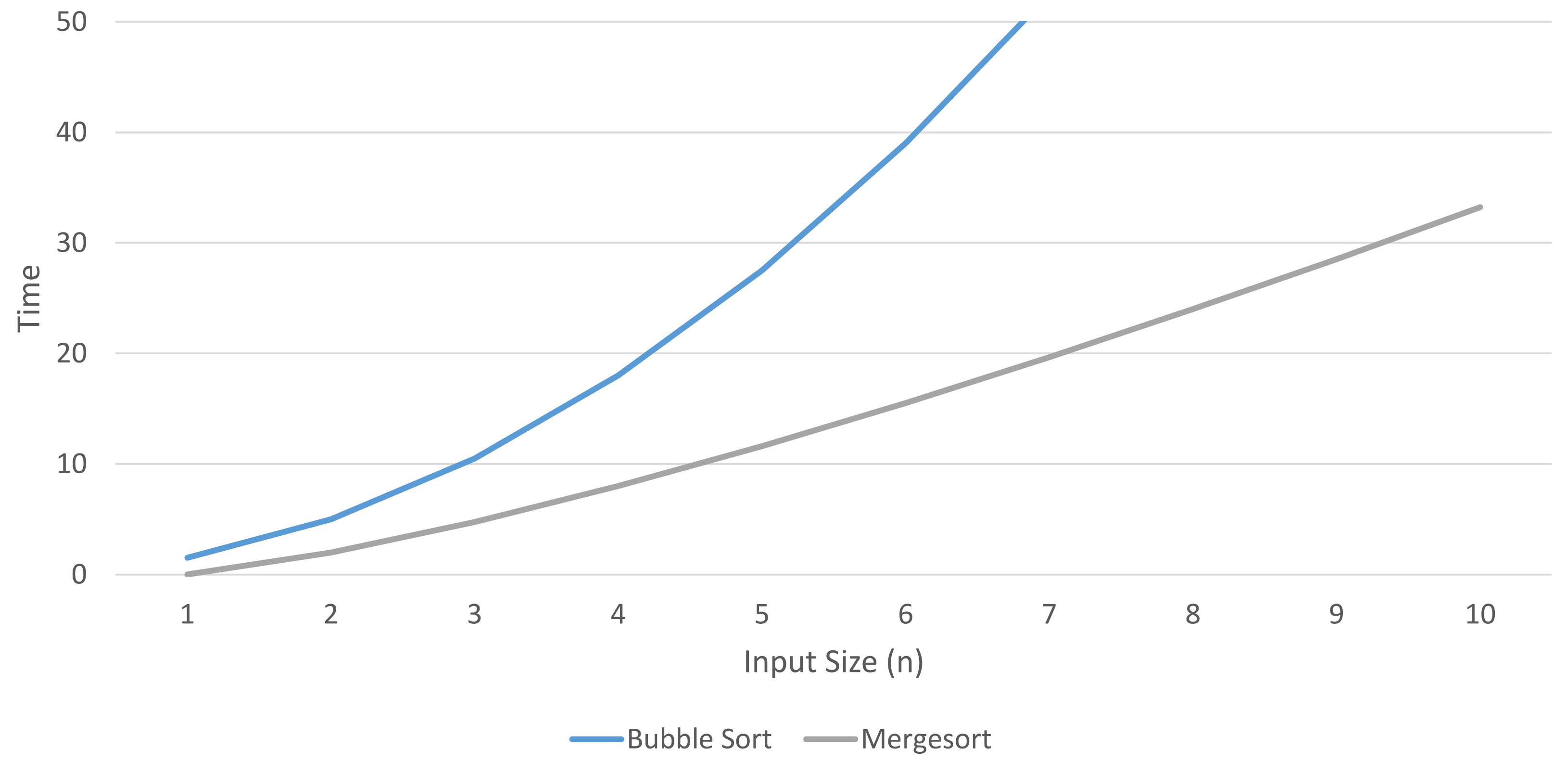
- Quasi-Linear (or Log-Linear) time is when an algorithm executes n-number of operations, where each operation performs in logarithmic time.
  - $T(n) = n \log_2 n$



- $T(n) = n \log_2 n$
  - $T(4) = 4 \log_2 4$
  - $T(8) = 8 \log_2 8$
  - $T(12) = 12 \log_2 12$

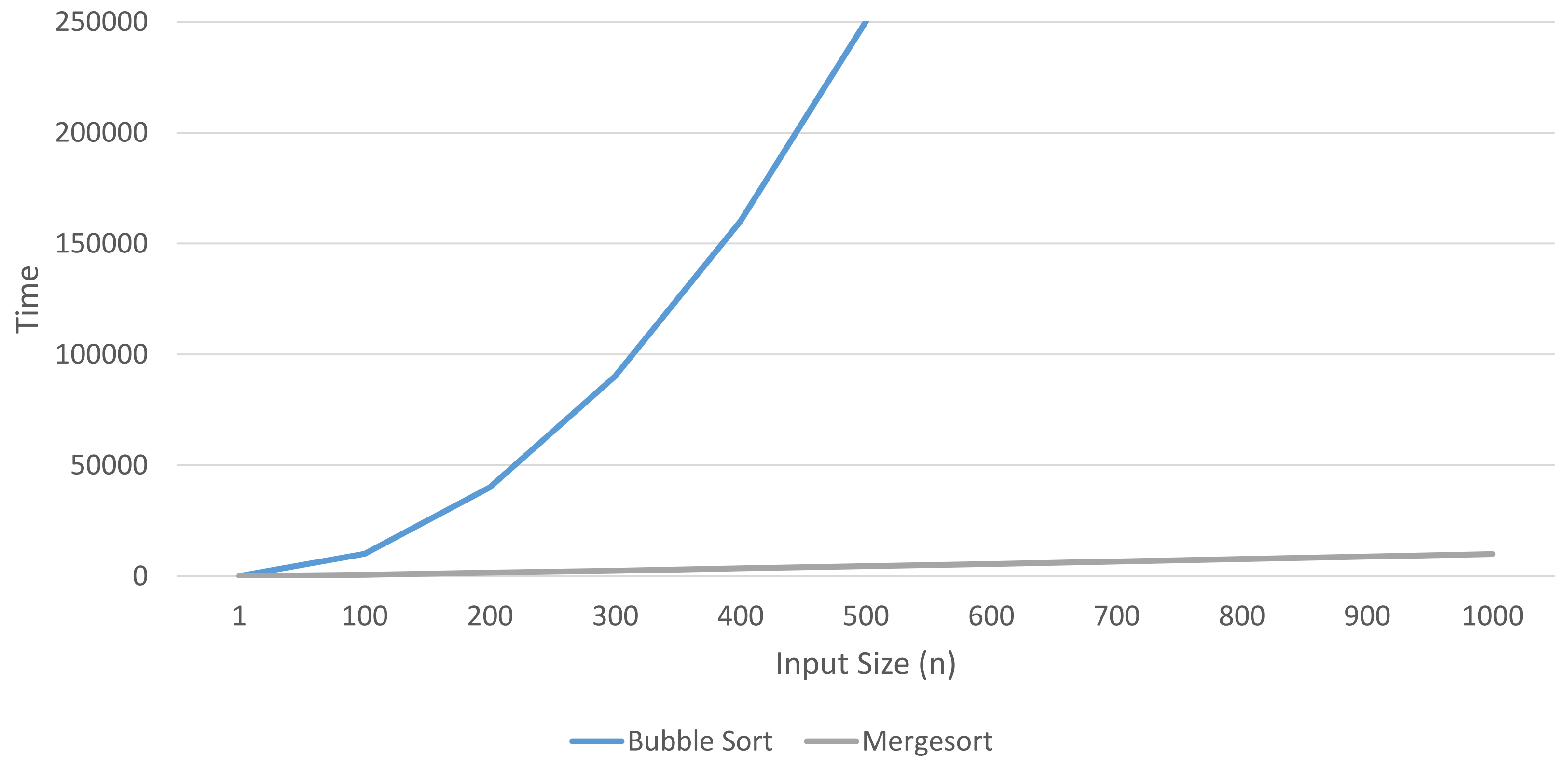
Bubble Sort vs Merge Sort

- Max array length: 10

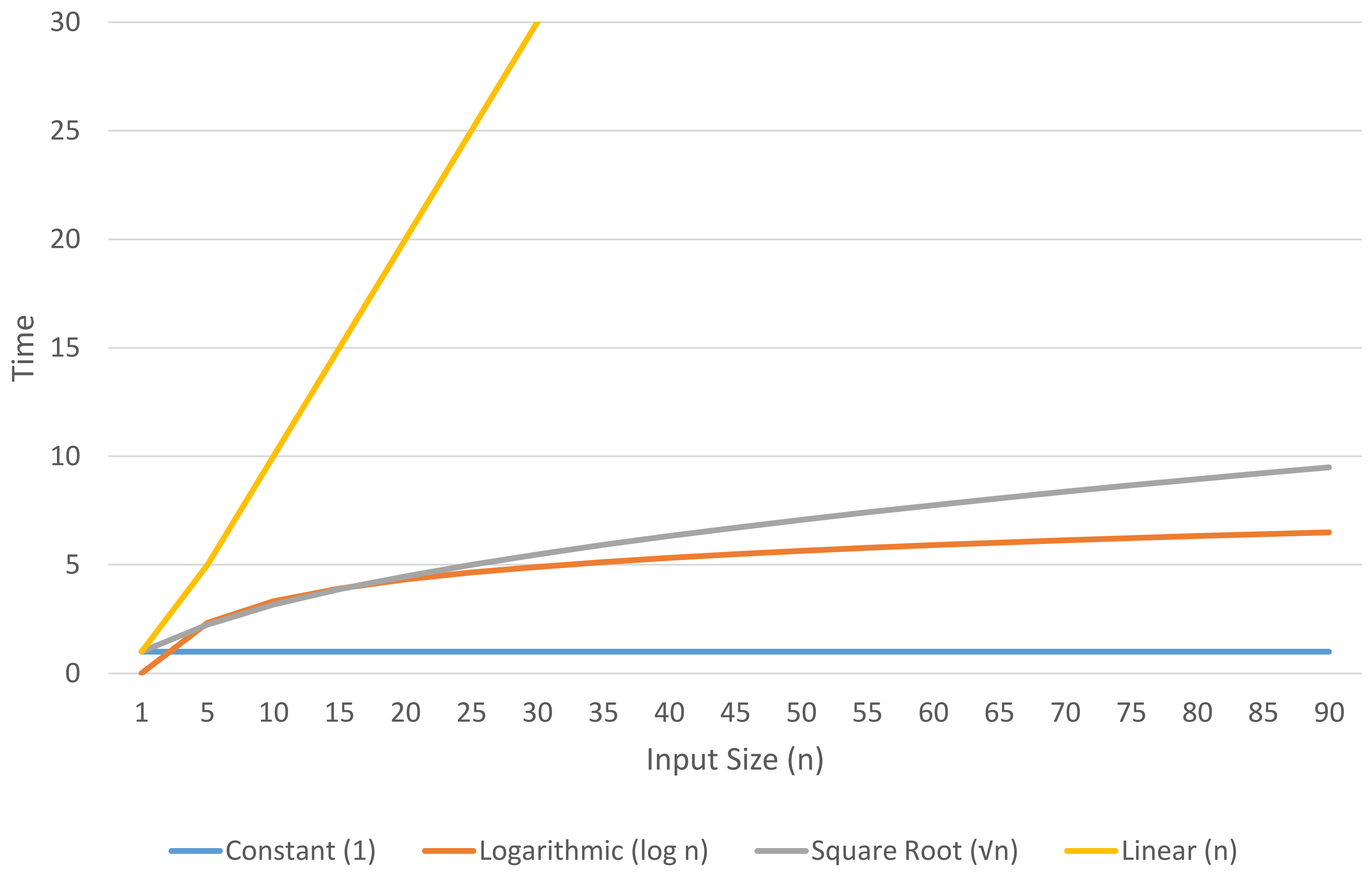


Bubble Sort vs Merge Sort

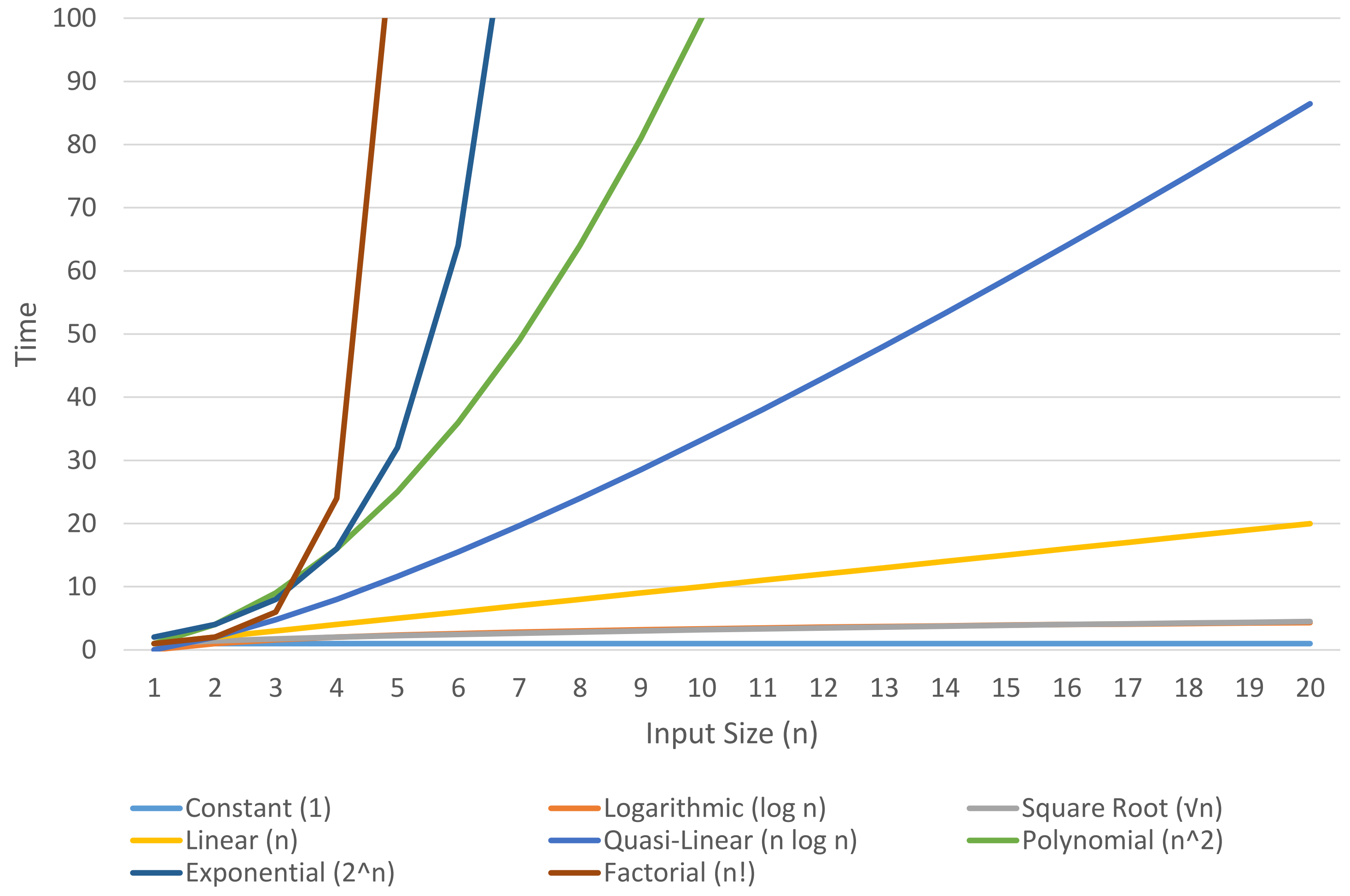
- Max array length: 1000



Complexities



Complexities


Time
Input Size (n)
Constant (1)
Logarithmic (log n)
Square Root (√n)
Linear (n)
Quasi-Linear (n log n)
Polynomial (n^2)
Exponential (2^n)
Factorial (n!)