Lab 03 Report

Mohamed Soliman

Here is the results of the profiling experiments for the fib and sort functions:

First: fib functions

fib1(): This function calculates the Fibonacci sequence values of a number n using recursion. The base case for this function is that it returns 1 if n = 1 or n = 2. To calculate the Fibonacci value for any other number, it uses recursion by calculating fib1(n-2) + fib1(n-1). This way, the function has a time complexity of $O(2^n)$, which is very inefficient and takes so much time. It also won't even work with very large numbers.

fib2(): This is an improved version of the fib1() function. It uses a dictionary to store the values of the Fibonacci sequence for each number. This way, it doesn't have to calculate the Fibonacci value for the same number more than once. This makes the function much more efficient than the fib1() function. It also works with larger numbers.

Second: sort functions

selectionSort(): This function sorts a list of numbers using the selection sort algorithm. It has a time complexity of $O(n^2)$ and is not very efficient. The way it works is that it goes through the list and finds the smallest number, then swaps it with the first number. Then it goes through the list again and finds the second smallest number and swaps it with the second number, and so on.

quickSort(): This function sorts a list of numbers using the quick sort algorithm. It has a time complexity of O(nlogn) and is much more efficient than the selection sort algorithm. The way it works is that it picks a pivot number and then puts all the numbers smaller than the pivot to the left and all the numbers larger than the pivot to the right. Then it does the same thing for the left and right sides of the pivot, and so on.