3500 hw 2

Casey Provitera

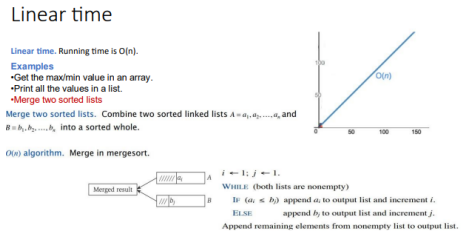
January 2024

1 (20 points) Question 1

You are given an array of integers. Write an algorithm with linear time complexity to check if this array satisfies the max heap property. The max heap property states that for any given node i, the value of array[i] should be greater than or equal to the values of its children. Your algorithm should return True if it's a valid max heap and False otherwise. Analyze the time complexity of your algorithm.

Algorithm Code	Time Complexity
1. checkHeap(A)	Total = n
2. for (int i=0, i<A.length, i++)	n
3. if(A[2i+1])	1
4. if(A[i] < A[2i] || A[i] < A[2i+1])	1
5. return False	1
6. return True	1

2 (30 points) Question 2



In class, we talked about the linear time complexity O(n), and we gave an example of merging two sorted lists into a single sorted list.

Now, let's elevate the complexity a bit. Imagine you have k sorted lists with n elements in total. Your task is to merge these k sorted lists into one single sorted list efficiently.
Here's the challenge: Your algorithm should accomplish this in O(n log k) time complexity. I'll offer a hint: use a heap for k-way merging.

In psudo-code	Time Complexity
mergeKSortedLists(k number of lists)	Total = n log k
OriginalList will be used as an attribute that elements have saying what list they were in at the start	
1. for list in k	k number of lists
2. middlestep[list] = (list[1], OriginalList)	log k
3. while middle step is not empty	n
4. minheap(middlestep)	log k
5. toRemove = middlestep.removeMin	log k
6. final.append(toRemove)	1
7. if toRemove.OriginalList is not empty	1
8. middlestep.root = (OriginalList[1], OriginalList)	1
9. OriginalList.reomve(index 1)	k
10. if middlestep is empty return final	1

3 (50 points) Question 3

In this segment of the assignment, your objective is to put into practice two algorithms designed to determine whether a given number can be expressed as the sum of two other numbers within a provided list of integers. To simplify matters, we'll allow the same number to be used twice when composing the sum. To illustrate, if the list contains the number 10, then 20 = 10 + 10 and thus 20 is considered to be a sum of two numbers from the list.

You need to implement two algorithms:
Brute-force: You simply try all possible pairs of numbers to see if a given number matches each sum.
Binary-search: You would first sort the list of numbers and then perform binary search on the numbers. You can use existing functions to perform sorting that is provided by the language. If you write sorting yourself, make sure to write an efficient sorting algorithm (O(nlogn) time).

Now implement both algorithms using your favorite language (perhaps Python). Then run your program on the provided data to compare how the two different algorithms perform on small to large data files. Here are more detailed instructions:
You have two sets of files:

1. files listNumbers-n (listNumbers-10, listNumbers-100, listNumbers-1000, listNumbers-10000, listNumbers-100000, listNumbers-1000000) each one of these files contain different data size 10, 100, 1000, 10000, 100000, 1000000 random numbers respectively.

2. files listNumbers-n-nsol, each one of these files contain 10 numbers.

What you need to do:

1. For each number x in listNumbers-n-nsol file:

2. look for two numbers in listNumbers-n file, where the sum of these two numbers = x , or one number if you use it twice will also sum to x (ex: x = 30 , and you found 15 in listNumbers-n list)

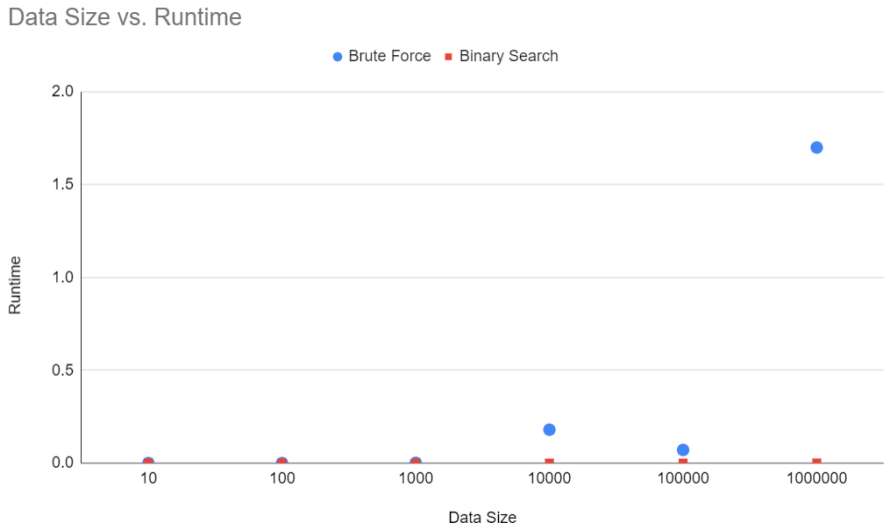
3. repeat that for all pair of files (listNumbers-n and listNumbers-n-nsol)

Note: you don't need to find all pairs that sum to x, if one found, then stop and proceed to the next number in list listNumbers-n-nsol
File listNumbers-10-wsol is an example of that.

What to submit?
Submit a short report containing the following.
Settings:
(10 points) Write down the language you use, the machine (its CPU frequency and memory size) you use for testing your program.
Language - pyhton
Machine CPU frequency - Processor 12th Gen Intel(R) Core(TM) i7-1260P, 2100 Mhz, 12 Core(s), 16 Logical Processor(s)
Machine memory size - 32 gigs of ram, 934 total gigs of storage
Results:
(10 points) Provide a table comparing the two algorithms by listing the average running time for each data size.

(10 points) Then plot the running time in a diagram (with the x-axis being the data size and y-axis being the running time); choose the proper scale to best demonstrate the results.

Data Size (x-axis)	Average Run Time in Seconds (y-axis)	
	Brute Force	Binary Search
10.00	9.9*10^-5	1.5*10^-4
100.00	1.9*10^-4	1.0*10^-4
1000.00	1.4*10^-3	2.0*10^-3
10000.00	1.8*10^-1	2.0*10^-3
100000.00	7.1*10^-2	2.2*10^-2
1000000.00	1.70	7.5*10^-2



(10 point) Conclusion: Draw conclusion on why choice of algorithm matters.
Algorithm choices clearly matter and it is especially important when using large data sizes with tedious tasks. The method that uses binary search is exponentially better than the brute force method on large data sizes. This is because the brute force method does not take many factors into account and simply sums every pair of numbers in hope they will add to a target value. The binary search method will take a target value and subtract a number from it then search for the difference in an effective way. Although the binary search is faster on large data sizes it does take a little bit more overhead memory which results in it only being slightly faster and sometimes slower than the brute force method on small data sizes.

(10 points) Code: Attach the source code of your implementation. If it is short enough, you may simply include your code as part of your report.
Source code will be in another file.