CS320 Homework 1

Dustin Randall

October 9, 2025

1 Problem 1: Describe the similarities and differences between data structures and algorithms.

Algorithms and data structures are fundamentally linked concepts, and play very different roles. A data structure represents the state of a system, while an algorithm uses state or states to produce another state. Said another way, algorithms take data structures as input, and produce data structures as output. Using an analogy from language, data structures are nouns, and algorithms are verbs.

2 Problem 2: Show that $5000n^2 + n \log n = O(n^2)$

Prove $\exists\, c > 0, n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ where $n \geq n_0$

$$5000n^2 + n \log n \leq cn^2 \tag{1}$$

$$5000 + \frac{\log n}{n} \leq c \tag{2}$$

Given that $\frac{\log n}{n}$ tends toward 0 as n increases, we know that the left side of the equation tends towards 5000.
Let us prove at least one valid c and n_0

let $\begin{cases} n_0 & = 100, \\ c & = 10\ 000 \end{cases}$

$$5000 + \frac{\log 100}{100} \leq 10\ 000 \tag{3}$$

$$5000 + \frac{2}{100} \leq 10\ 000 \tag{4}$$

$$5000 + 0.02 \leq 10\ 000 \tag{5}$$

$$5000.02 \leq 10\ 000 \tag{6}$$

3 Problem 3: Show: $5000n^2 + n\log n = o(n^3)$

Prove for every positive constant $c > 0$, $\exists\, n_0 > 0$ such that $0 \leq f(n) < cg(n)$ where $n \geq n_0$

$$5000n^2 + n\log n < cn^3 \tag{7}$$

$$\frac{5000}{n} + \frac{\log n}{n^2} < c \tag{8}$$

Given that $\frac{5000}{n}$ and $\frac{\log n}{n^2}$ both tend toward 0 as n increases, the entire left side tends toward 0. Thus, for any positive c we can find an n_0 such that the equation will hold true.

4 Problem 4: Show: $5000n^2 + n\log n = \Omega(n^2)$

Prove $\exists\, c > 0, n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ where $n \geq n_0$

$$cn^2 \leq 5000n^2 + n\log n \tag{9}$$

$$c \leq 5000 + \frac{\log n}{n} \tag{10}$$

Given that $\frac{\log n}{n}$ tends toward 0 as n increases, let us prove at least one valid c and n_0

let $\begin{cases} n_0 & = 10, \\ c & = 1\ 000 \end{cases}$

$$1\ 000 \leq 5000 + \frac{\log 10}{10} \tag{11}$$

$$1\ 000 \leq 5000 + \frac{1}{10} \tag{12}$$

$$1\ 000 \leq 5000 + 0.1 \tag{13}$$

$$\tag{14}$$

5 Problem 5: Show: $5000n^2 + n\log n = \omega(n)$

Prove for every positive constant $c > 0$, $\exists\, n_0 > 0$ such that $0 \leq g(n) < f(n)$ where $n \geq n_0$

$$cn < 5000n^2 + n\log n \tag{15}$$

$$c < 5000n + \log n \tag{16}$$

Given that $5000n$ tends toward infinity as n increases, we can find an n_0 such that the equation will hold true for any positive c.

6 Problem 6: Given f(n) is a positive increasing function, prove or contradict: $f(n) = \Theta(f(2n))$

Given that $f(n)$ is positive and increasing, we can say that $f(2n) > f(n)$. In the inequality $c1f(2n) \leq f(n) \leq c2f(2n)$, we know that the right side is true for all $c2 \geq 1$. This leaves us to prove the left side.

$$c1f(2n) \leq f(n) \tag{17}$$

$$c1 \leq \frac{f(n)}{f(2n)} \tag{18}$$

In order for this to be true, $\frac{f(n)}{f(2n)}$ must have a lower bound greater than 0. If we try something like $f(n) = n$ we can show $\frac{n}{2n} = \frac{1}{2}$ which has a lower bound greater than 0. However, if we use $f(n) = 2^n$ we can see that $\frac{2^n}{2^{2n}} = \frac{2^n}{(2^n)^2} = \frac{1}{2^n}$. This approaches 0 as n increases, thus there is no constant guaranteed to be less than the lower bound. Said more intuitively, given that $\Theta(f(n))$ is dependent on the largest term, if n is in the exponent, then doubling n will increase the growth rate of $f(n)$. Another counter-example is $f(n) = n!$.

$$c1 \leq \frac{n!}{(2n)!} \tag{19}$$

$$c1 \leq \frac{(n)(n-1)(n-2)\dots}{(2n)(2n-1)\dots(n)(n-1)(n-2)\dots} \tag{20}$$

$$c1 \leq \frac{1}{(2n)(2n-1)\dots(n+1)} \tag{21}$$

This series also approaches 0 as n increases.

7 Problem 7: Evaluate a merge sort with subsets of size 4 instead of 2.

7.1 Pseudocode

```
1  WideMerge(arr, start, end)
2      len = end - start + 1
3      if(len < 2) return
4      if(len < 4)
5          Merge(arr, start, end)
6          return
7      groupSize = len / 4
8      mid1 = start + groupSize
9      mid2 = mid1 + groupSize
10     mid3 = mid2 + groupSize
11     WideMerge(arr, start, mid1)
```

```
12      WideMerge(arr, mid1, mid2)
13      Combine(arr, start, mid2)
14
15      WideMerge(arr, mid2, mid3)
16      WideMerge(arr, mid3, end)
17      Combine(arr, mid2, end)
18
19      Combine(arr, start, end)
```

7.2 Evaluate Θ

WideMerge = $T(n)$
Combine = $\Theta(n)$
WideMerge calls itself 4 times, each with a quarter of the array: $T(\frac{n}{4})$
WideMerge calls Combine three times, twice with $\frac{n}{2}$ and once with n: $c \cdot 2n$
This give the recurrence equation:
$T(n) = 4T(\frac{n}{4}) + c \cdot 2n$
We know in the base case that $T(1) = 1$, a constant time operation.
Using the branching tree approach:

$$T(n) = 4T(\frac{n}{4}) + c \cdot 2n \tag{22}$$
$$T(\frac{n}{4}) = 4T(\frac{n}{16}) + c \cdot \frac{n}{2} \tag{23}$$
$$T(\frac{n}{16}) = 4T(\frac{n}{64}) + c \cdot \frac{n}{8} \tag{24}$$
$$\vdots \tag{25}$$
$$T(1) = 1 \tag{26}$$

We know that each row of the tree has a cost of $c \cdot 2n$ times the number of rows which is $\log_4 n$.
After dropping the leading constants, this provides us with the time complexity of $\Theta(n \log n)$. There may be a way to reduce the number of Combine operations for 3 to 1 with additional conditional checks, but that would not change the overall time complexity.

7.3 Is the runtime asymptotically faster than traditional merge sort?

Both algorithms have the same time complexity: $\Theta(n \log n)$. While the constants may be different, the very definition of Θ means that two functions with the same Θ have the exact same asymptotic growth rate.

7.4 Could this algorithm be faster in practice?

There may be input sets which this algorithm performs better on, such as when the input is so large that a $\log_2$ approach leads to a stack overflow. Given that

the Combine function is where the majority of the time is spent, reducing it from 3 calls to 1 may result in an improvement as well, especially if written in a branchless way.

7.5 Can a subset size of 1 be asymptotically faster than traditional merge sort?

Using a subset of size n effectively performs the same algorithm without recursion.
Instead of using stack space to keep track of all the starts, middles, and ends, those become local variables. Because of this, the time complexity still cannot be faster than $\Theta(n \log n)$, but it can certainly be slower. As another justification, if someone had discovered a sorting algorithm faster than $\Theta(n \log n)$ it would render nearly every other sort obsolete.

7.6 Why does the text use subset of size 2?

I believe that using a subset size of 2 best illustrates the algorithm, with the fewest lines of code. All things begin equal, simple is better, and given that Θ of any type of merge sort are equal, the simplest implementation is the best.

8 Problem 8: Learn and describe 3 divide-and-conquer algorithms.

8.1 Quick Sort

8.1.1 The problem statement

Given an array of elements, sort the elements in non-decreasing order.

8.1.2 The idea

This algorithm works by partitioning the array into two chunks around a selected pivot element. Essentially $arr = [\text{elements} < \text{pivot}] + [\text{pivot}] + [\text{elements} \geq \text{pivot}]$
Much of the efficiency of this algorithm comes from defining a good pivot strategy.

8.1.3 Time complexity

The worst case time complexity is $O(n^2)$, the best case is $O(n \log n)$

8.1.4 Worst-case condition

The worst case occurs when the pivot strategy always selects the smallest or largest element, as the partitions will only shrink by 1 element per iteration. For example, if the array is all duplicated elements.

8.1.5 Sources

- https://www.csestack.org/quicksort/

8.2 QuickHull

8.2.1 The problem statement

Given a set of points in a 2D plane, find a polygon which encloses all the points, with minimal area.

8.2.2 The idea

This algorithm works similarly to Quick Sort. It sorts all the points by the x coordinate, as the first and last points are guaranteed to be part of the result. From there, a line is drawn between the two points, and 2 triangles are made. The third part of each triangle is the furthest point from the 2 points in the line. Any points inside the triangle can be discarded, and the remaining points are divided into two new sets. This is repeated recursively until the hull is complete.

8.2.3 Time complexity

The time complexity is similar to Quick Sort. The worst case is $O(n^2)$, and I believe the average case is $O(n \log n)$.

8.2.4 Worst-case condition

This algorithm's degenerate case is when all points are on the hull, and none can be discarded.

8.2.5 Sources

- https://www.gorillasun.de/blog/quickhull-algorithm-for-convex-hulls/
- https://en.wikipedia.org/wiki/Quickhull

8.3 Voronoi Diagram

8.3.1 The problem statement

Given a set of points in a 2D plane, divide the plane into regions where each region contains all the points which are closest to the given point.

8.3.2 The idea

This algorithm works by dividing the set of points in half, and recursively finding the Voronoi diagram for each half. The halves are then merged together using the perpendicular bisector of each of the edges between left and right halves.

8.3.3 Time complexity

The time complexity for the divide and conquer implementation is $n \log n$. This is bounded by sorting the points in $n \log n$ time, and merging which can be either $O(n)$ or $O(\log n)$.

8.3.4 Worst-case condition

Given that the input is bounded by the sort step, the worst-case condition is the same as the worst-case for the sorting algorithm.

8.3.5 Sources

- http://personal.kent.edu/ rmuhamma/Compgeometry/MyCG/Voronoi/-DivConqVor/divConqVor.htm