Divide & Conquer
Problem Set 1 – CS6515 (Spring 2025)

- This problem set is due on **Thursday January 16th**.
- Submission is via Gradescope.
- Your solution must be a typed pdf (e.g., via LaTeX, Markdown, etc. Anything that allows you to type math notation) – no handwritten solutions.
- Please try to make your solutions as concise and readable as possible. Most problems will have solutions that are no more than a page long. Consider using bullet points and adding space to break up large paragraphs into smaller chunks.
- There are 3 problems. Each problem is graded with 20p. **There is +1p bonus per problem** for stating (i) how long it took you to solve that problem, and (ii) how long it took you to type the answer.

Remark on algorithm descriptions To make things easier for the TAs to grade, please use a combination of plain English and mathematical notation. Do not write code (C, Java, etc.). For example, you can say something like "$x := \max_{i=0,\dots,n-1}$ `A[i]`" or "Find the maximum value in the array A" instead of writing a for-loop that computes the maximum.

1 Complexity Analysis I

Find an accurate bound $T(n,k) = O(...)$.

1. Bound $T(n,k) = k \cdot T(n/2, k) + 2^k n^2$ with base case $T(1,k) = 2^k$. Consider all cases where $1 \le k \le n$.

2. Ask ChatGPT (or Gemini, Copilot, AppleAI, etc.) to solve this problem. Copyable text:
   `Bound $T(n,k) = k \cdot T(n/2, k) + 2^k n^2$ with base case $T(1,k) = 2^k$. Consider all cases where $1 \le k \le n$.`
   Copy its answer and grade it (i.e., mark the mistakes and briefly explain why they're wrong).

2 Complexity Analysis II

1. Give an accurate bound $O(...)$ for $T(n) = T(n/2) + T(n/3) + O(n)$.

2. Prove that $\log(n!) = O(n \log n)$ and $\log(n!) = \Omega(n \log n)$.

You may **not** use Sterling approximation for (2), as that trivializes the exercise. Instead, work directly with the definition $n! = n \cdot (n-1) \cdot (n-2) ... \cdot 2 \cdot 1$ and use $\log ab = \log a + \log b$.

3 Divide&Conquer Algorithm

Consider the following "maximum-quality" problem:

Maximum-Quality We are given access to some function $Q(\cdot)$ and two integers $m \leq n$. For integers i, j, the function $Q(i, j)$ returns some "quality". We are promised that the function satisfies $Q(i, k) + Q(k, j) = Q(i, j)$ for all $i \leq k \leq j$. Calling/executing the function takes $O(1)$ time.

The task is to find indices i, j with $m \leq i \leq j \leq n$ where $Q(i, j)$ is as large as possible.

Problem Construct a divide&conquer algorithm $\textsc{MaxQuality}(Q, m, n)$ that solves the above problem. Your submission must provide the following:

(a) A description of your algorithm.

(b) A correctness argument. You may provide a proof by induction, but answering the following questions also suffices:

  (b,i) Why is the base case correct (i.e., when $m = n$)?

  (b,ii) Why does the algorithm return a correct answer, if it has the solution for the left and right half of the problem?

(c) Complexity analysis (e.g., via Master Theorem).

Remark/additional background: As motivation for the maximum-quality problem, consider the following two problems. They are common dynamic programming exercises, but today we want to solve them via a single divide&conquer algorithm.

Maximum Profit We are given an array $A[0...n-1]$ where entry $A[i]$ represents the price of some object at time i. We want to buy the object at a low cost, and later sell it again at a high cost. We want to maximize the profit, i.e., find indices $i \leq j$ with the largest $A[j] - A[i]$.

If we let $Q(i, j) = A[j] - A[i]$, then maximum-profit is solved via the maximum-quality problem.

Maximum Subarray Sum We are given an array $A[0...n-1]$ and must find the maximum possible sum of a *contiguous* subarray, i.e., find $i \leq j$ where the sum $\sum_{k=i}^{j} A[k]$ is as large as possible.

If we let $Q(i, j) = \sum_{k=i}^{j-1} A[k]$, then maximum-subarray-sum is solved via the maximum-quality problem.

So both of these common DP problems are special cases of the maximum-quality problem.