

Problem Set 2: Divide and Conquer **Sketch**

Released 07 February 2024

Due 14 Feb 2024 10:30pm in Gradescope

Instructions. You may discuss the material with your classmates, but you must write solutions yourself. List collaborators on your submission. Do not use other sources of help other than the course staff and the textbook.

If you are asked to design an algorithm, please provide: (a) the pseudocode or precise description in words of the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submissions. Please submit a PDF file typed using L^AT_EX. Please assign pages to questions in Gradescope. **TODO: Solutions 2,5(see notes at home),Bonus**

1. 4.3-1ab Substitution Method.

Solution:

- a: $T(n) = T(n-1) + n$ has solution $T(n) = O(n^2)$

Since our goal is to show $T(n) = O(n^2)$, let us assume that $T(n) \leq cn^2$ for all values as least as large as n_0 and less than n .

$$\begin{aligned}
 T(n) &= T(n-1) + n \\
 &\leq c(n-1)^2 + n - 1 + n \\
 &= cn^2 - 2cn + c + 2n - 1 \\
 &= cn^2 + (c-1)(1-2n) \\
 &\leq cn^2
 \end{aligned} \tag{1}$$

(for sufficiently large c, n_0)

- b: $T(n) = T(n/2) + \Theta(1)$ has solution $T(n) = O(\log n)$

Since our goal is to show $T(n) = O(\log n)$, let us assume that $T(n) \leq c \log n$ for all values as least as large as n_0 and less than n . In particular if $n \geq 2n_0$ it holds for $\frac{n}{2}$

(for sufficiently large c, n_0 , $\log 2$ dominates $\Theta(1)$)

Solution:

2. 4.4-1ac Recursion Tree analysis. Just sketch the tree and give a reasonable guess on the upper bound. What is the height, what is the cost of the first, second, and last level?

Solution:

- a: $T(n) = T(n/2) + n^3$
 - Guess: n^3
 - Height: $\log_2 n$
 - Cost first: n^3
 - Cost second: $\frac{n^3}{2}$
 - Cost last: $\Theta(1)$
 - Substitution (not required): $T(n) \leq \frac{cn^3}{8} + n^3 \leq cn^3$ if $c > 2$ so $T(n) = O(n^3)$

3. 4.5-1abde (no c) Tight Asymptotic Bounds (Θ)

Solution: In all cases we have that $a=2$, $b=4$, so $n^{\log_b a} = n^{\log_4 2} = \sqrt{n}$

- a: $T(n) = 2T(n/4) + 1$. Case 1:
- b: $T(n) = 2T(n/4) + \sqrt{n}$. Case 2:
- d: $T(n) = 2T(n/4) + n$. Case 3:
- d: $T(n) = 2T(n/4) + n^2$. Case 3:

4. 4.5-3 Binary Search Analysis

Solution: $T(n) = T(n/2) + \Theta(1)$

$a=1$, $b=2$

$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 = \Theta(1)$

Solution:

5. Max Give an efficient algorithm for finding the maximum value of an array of size n (assume n is a power of 2) using divide and conquer. Prove that it is correct, and argue the runtime. Is this any faster than an iterative approach using a for loop to search every element in order?

Solution: Sketch: Tournament

Algorithm 1: MaxArray(A, n)

Input: Array A of size n

Output: maximum element

```

1 //Base Case if Len(A)==1 then
2   return (A[0])
3 else
4   //Divide
5   Left=MaxArray([1:n/2],n/2)
6   Right=MaxArray(A[n/2+1:n],n/2)
7   //Conquer
8   return max(Left,Right)
```

Correctness: This function recurses through the array until it hits each individual element. It then compares the elements in a tournament style to determine the maximum. The maximum element of the array will be greater than (or equal to) all other elements, so it doesn't matter in what order it is compared to others. (This would not work if we wanted the second largest element for example)

Runtime: In each step, the size of the problem is divided into two sub problems of size $n/2$. Taking the max of two elements is constant time. Therefore we have the recurrence relation: $T(n) = 2T(n/2) + \Theta(1)$. By the master theorem this is still $O(n)$. **Which is no faster than the iterative approach**

Bonus (token opportunity). 4-6 Chip Testing page 122 (hardcover).

Solution:

- (a) Bad chips will claim bad chips are good and good chips are bad (inverse of what the good chips do). If at least half of the chips are bad, then there is no way to distinguish a good chip from a bad chip.
- (b) Do $n/2$ pairwise comparisons and keep the pairs where both claim the other is good. Since more than half the chips are good, we know that more than half of such pairs will be pairs of good chips. Since more than half the chips are good, we also know that there will be at least one such pair. Arbitrarily pick one chip from each pair. This cuts the size of the problem to at most $\lceil n/2 \rceil$, and retains the fact that more than half of the chips are good. *Edge case, if there are an odd number of chips, keep the chip that is unpaired.
- Recursively apply until three chips are remaining. Since there are more good chips than bad chips, we have either GGB or GGG. Check any pair. If both say the other chip is good, arbitrarily keep one. Otherwise keep the extra chip and discard both of the others. This chip will be good. $T(n)=T(n/2)+n/2$ (in each step we do $n/2$ constant time comparisons and cut the problem in half). Using the master theorem, we find that $T(n) = \Theta(n)$
- To determine the rest of the good chips do n pairwise comparisons between the good chip and all other chips. If Chip A (the good chip) says that the other chip is good, it is good, otherwise it is bad.