

Week 6 Day 1 MIDTERM REVIEW

Prep:

- Midterm Review Problems posted
- Solutions to problems are also posted.
- Review Quiz answers.

Integral Scheduling: (Greedy Algorithm Slides)

- Greedy Algorithm: Solving using the simplest algorithm to implement.
- Doesn't guarantee optimal runtime but can be useful.

Greedy Algorithm for Heuristics:

Option 1: Select earliest starting task

Option 2: Select the Shortest Available Task

Option 3: Schedule task with fewest conflicting tasks

- **All 3 of these greedy options don't guarantee an optimal solution**
- Come up with examples that fail with these criteria.

Option 4: Earliest Finishing Time

Earliest Finish Time First (Algorithm Pseudo-Code)

- **Sort Tasks** in finish time order.

EARLIEST-FINISH-TIME-FIRST ($n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$)

Sort tasks by finish time so that $f_1 \leq f_2 \leq \dots \leq f_n$

$A \leftarrow \Phi$

for $i=1$ to n

 if task i is compatible with A

$A \leftarrow A \cup \{i\}$

Return A

Annotations:

- $O(n)$ { (next to the for loop)
- remove tasks that has conflict with selected ones (next to the if statement)

Runtime: $O(n \log n)$ - sorting

To check if task i is compatible with A , just need to keep track of the last added task j^* and compare s_i to f_{j^*}

Runtime Speed: $O(n \log n)$ – Sorting

To check for compatibility: Compare Current last finish time and new start time.

EARLIEST-FINISH-TIME-FIRST Optimal (**Analysis Picture***) EFTF

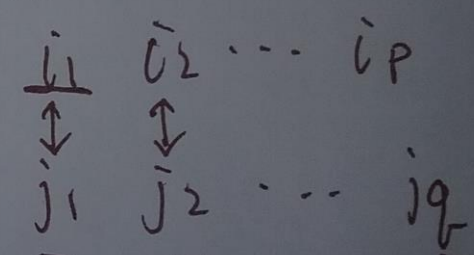
Analysis of EARLIEST-FINISH-TIME-FIRST

Claim: the EARLIEST-FINISH-TIME-FIRST algorithm is optimal

Key idea: EARLIEST-FINISH-TIME-FIRST stays ahead

- Let $i_1 i_2 \dots i_p$ be the set of tasks selected by greedy ordered by finish time
- Let $j_1 j_2 \dots j_q$ be a set of tasks selected by a different algorithm ordered by finish time

We can show that for $r \leq \min(p, q)$, $f(i_r) \leq f(j_r)$



The diagram illustrates the 'stays ahead' property. It shows two rows of task indices. The top row contains i_1, i_2, \dots, i_p and the bottom row contains j_1, j_2, \dots, j_q . Vertical double-headed arrows connect i_1 to j_1 , i_2 to j_2 , and i_p to j_q , representing the comparison of finish times at each step r .

Proof by Induction (See Stay Ahead Lecture Slides*)

J_{R+1} Must be compatible with i_{R+1} because of the definition of how the greedy algorithm chooses the next element. Therefore, the finish time for the $K+1$ step is no later than the J_{r+1} .

Completing the Proof:

The EFTF solution has to have an equal number of tasks as the optimal solution q .

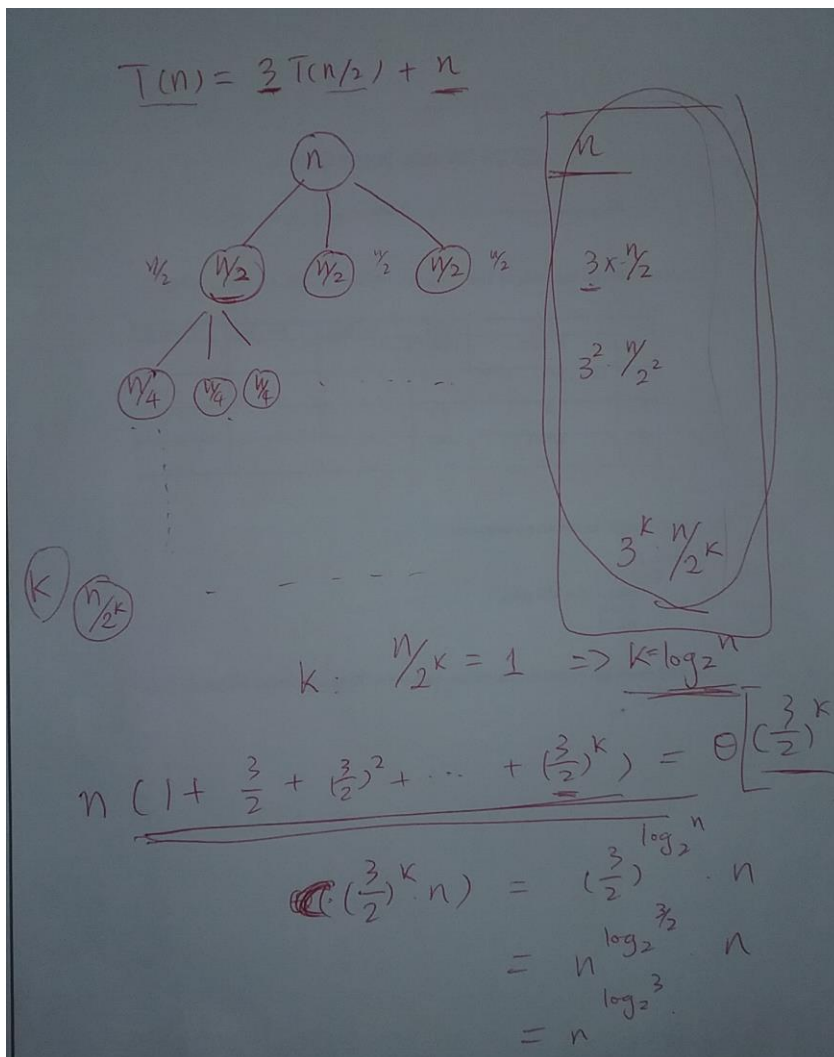
So p must be equal to q because if it's less than q then q is not the optimal solution.
 $p = q$. p cannot be less than q .

Midterm Review:

Asymptotic Runtime Analysis:

- Big O(\leq), Theta ($=$) Big Omega (\geq)
- Email of Asymptotic order strategies.
- $T(n) = T(n-1) + T(n-2) \leq 2T(n-1)$
- Rule of Thumb.
- Limit of Ratio. (Lim as $n \rightarrow \text{Infinity}$ of: $f(n) / g(n)$)
- Convert Pseudo-Code to Runtime.
- Recursion Tree for Solving Recurrence relations.
- Master Theorem.
- Geometric Series

Recursion Tree Example: (Picture)



Divide and Conquer:

- Break down into small sub-problems
- Solve them via recursion
- Combine solutions of sub-problems to get the original problem
- Example: merge sort ($n \log n$) break into half over and over. Spend linear time to combine them back.
- Binary search ($\log n$) Don't need to recombine.
- Merge Sort Recurrence Relation: ($T(n) = T(n/2) + C$)
- **Majority Element Problem: (from Midterm Review)**
- *High Level Idea Picture*

High level idea

- Break A into A_L and A_R
- We can recursively find the majority element in A_L and A_R - call them m_L and m_R
- For possible outcomes:
 1. $m_L = m_R = \text{NULL}$
 2. $m_L = m_R \neq \text{NULL}$
 3. $m_L \neq \text{NULL}, m_R = \text{NULL}$
 4. $m_L = \text{NULL}, m_R \neq \text{NULL}$
 5. $m_L \neq m_R$ both not NULL.
- Key insight: if an element is majority of A , it has to be majority for either A_L and A_R
 - Otherwise, its total occurrence would be $\leq n/2$

Proof by Induction:

- Base Case
- Inductive Assumption: Assume correct up to k .
- Inductive Step: Consider $k+1$

Dynamic Programming: (See Picture*)

- 1. Define Subproblems to solve.
- 2. Give recurrence relation for the Subproblems. Provide Base Case.
- 3. Give runtime of algorithm and explain.

- Review for **Midterm** on Thursday
- More Officer Hours will be available for Wednesday (check Email and Canvas for Dates)
- No Recitation Wednesday.
- **Continue Working on Assignment 2**

~Information composed by Notetaker Scott Russell for CS 325 **DAS** student