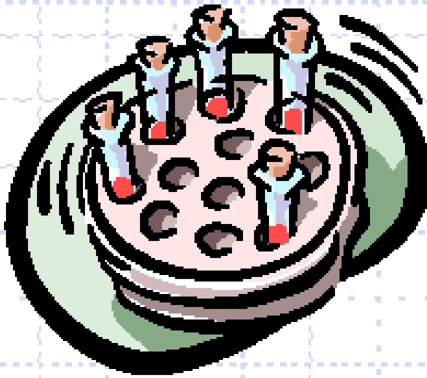
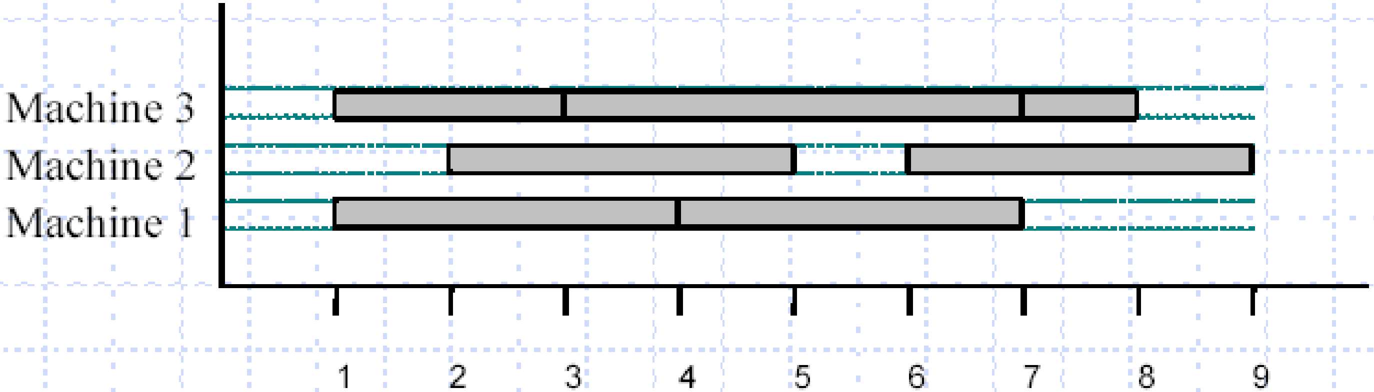
Task Scheduling, Optimal Merge Patterns

Task Scheduling



- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
- Goal: Perform all the tasks using a minimum number of "machines."



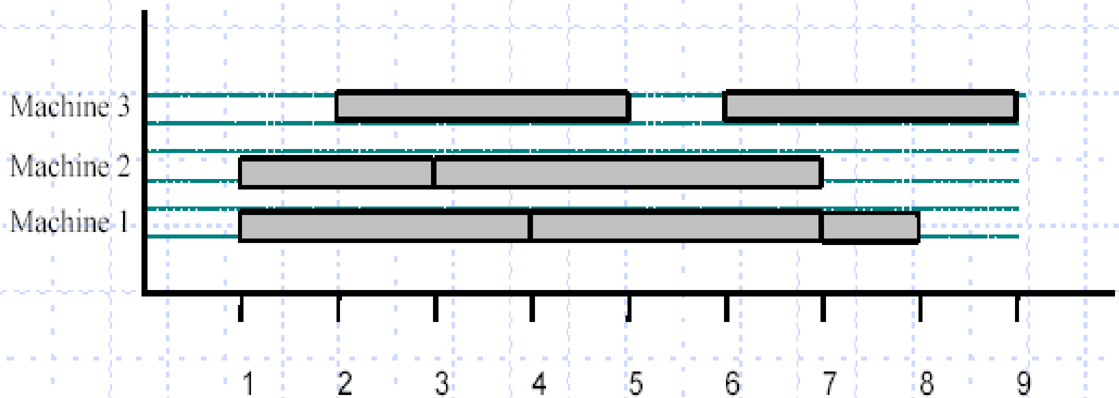
Task Scheduling Algorithm

- Greedy choice: consider tasks by their start time and use as few machines as possible with this order.
  - Run time: O(n log n). Why?
- Correctness: Suppose there is a better schedule.
  - We can use k-1 machines
  - The algorithm uses k
  - Let i be first task scheduled on machine k
  - Machine i must conflict with k-1 other tasks
  - But that means there is no non-conflicting schedule using k-1 machines

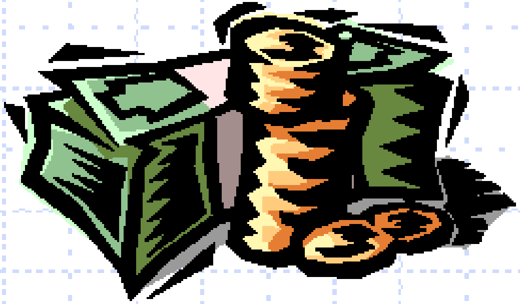
```
Algorithm taskSchedule(T)
    Input: set T of tasks w/ start time s_i
    and finish time f_i
    Output: non-conflicting schedule
    with minimum number of machines
    m ← 0                    {no. of machines}
    while T is not empty
        remove task i w/ smallest s_i
        if there's a machine j for i then
            schedule i on machine j
        else
            m ← m + 1
            schedule i on machine m
```

Example

- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
  - [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8] (ordered by start)
- Goal: Perform all tasks on min. number of machines



Making Change



- Problem: A dollar amount to reach and a collection of coin amounts to use to get there.
- Configuration: A dollar amount yet to return to a customer plus the coins already returned
- Objective function: Minimize number of coins returned.
- Greedy solution: Always return the largest coin you can
- Example 1: Coins are valued \$.32, \$.08, \$.01
  - Has the greedy-choice property, since no amount over \$.32 can be made with a minimum number of coins by omitting a \$.32 coin (similarly for amounts over \$.08, but under \$.32).
- Example 2: Coins are valued \$.30, \$.20, \$.05, \$.01
  - Does not have greedy-choice property, since \$.40 is best made with two \$.20's, but the greedy solution will pick three coins (which ones?)

Optimal 2-way Merge patterns

Suppose there are 3 sorted lists *L*1, *L*2, and *L*3, of sizes 30, 20, and 10, respectively, which need to be merged into a combined sorted list but we can merge only two at a time.

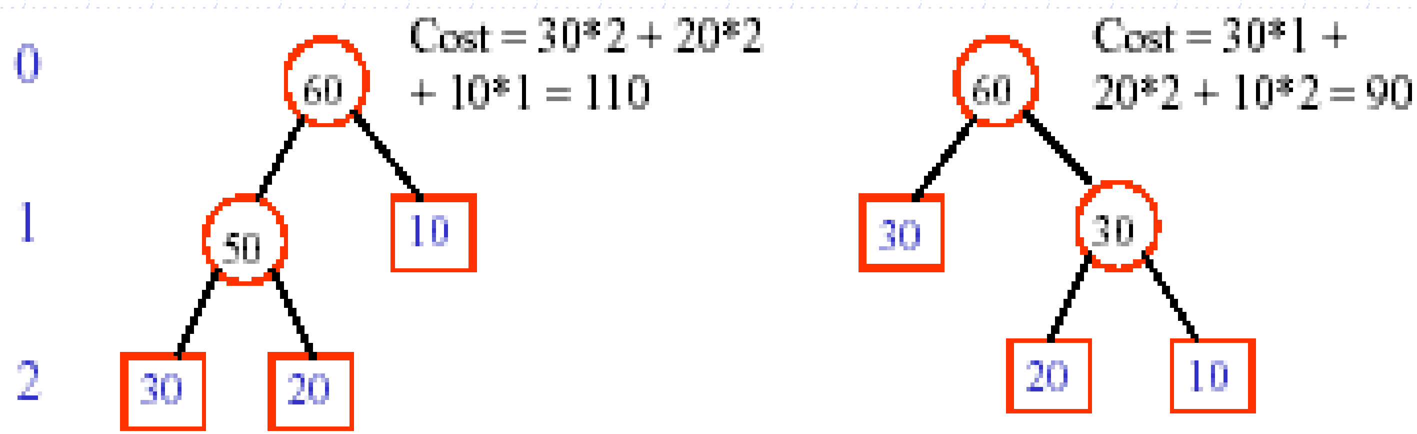
- We intend to find an optimal merge pattern which minimizes the total number of comparisons..

Example

- **merge *L*1 & *L*2,: 30 + 20 = 50 comparisons,then merge the list & *L*3: 50 + 10 = 60 comparisons**
- **total number of comparisons: 50 + 60 = 110.**
- **Alternatively, merge *L*2 & *L*3: 20 + 10 = 30 comparisons, the resulting list (size 30) then merge the list with *L*1: 30 + 30 = 60 comparisons**
- **total number of comparisons: 30 + 60 = 90.**

Binary Merge Trees

- using a binary tree, built from the leaf nodes (the initial lists) towards the root in which each merge of two nodes creates a parent node whose size is the sum of the sizes of the two children.



Merge L_1 and L_2, then L_3 Merge L_2 and L_3, then L_1

merge cost = sum of all weighted external path lengths

Optimal Binary Merge Tree Algorithm:

- Input: n leaf nodes each have an integer size, $n = 2$.
- Output: a binary tree with the given leaf nodes which has a minimum total weighted external path lengths

Algorithm:

(1) create a min-heap $T[1..n]$ based on the n initial sizes.

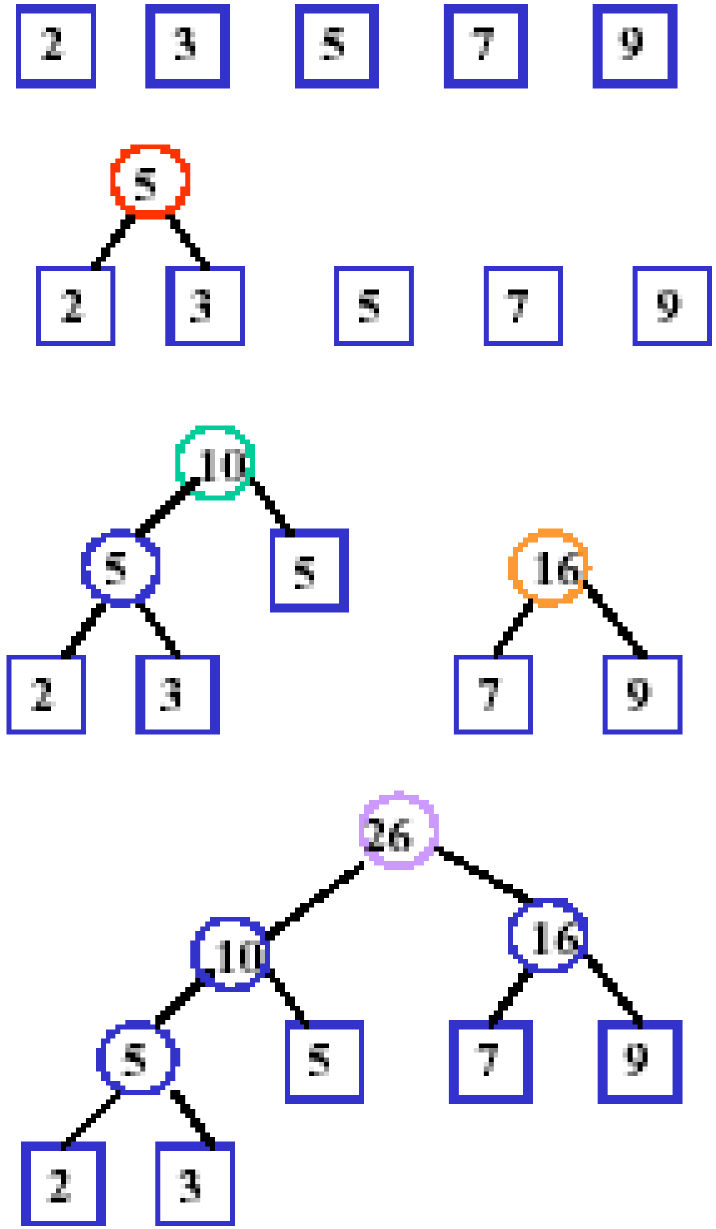
(2) while (the heap size >= 2) do

(2.1) delete from heap two smallest values, a and b, create a parent node of size $a + b$ for the nodes corresponding to these two values

(2.2) insert the value $(a + b)$ into the heap which corresponds to the node created in Step (2.1)

When algorithm terminates there is single value left in heap whose corresponding node is the root of the optimal binary merge tree.

- time complexity: $O(n \lg n)$: Step (1) takes $O(n)$ time; Step (2) runs $O(n)$ iterations, in which each iteration takes $O(\lg n)$



Initially, 5 leaf nodes with sizes

Iteration 1: merge 2 and 3 into 5

Iteration 2: merge 5 and 5 into 10

Iteration 3: merge 7 and 9 (chosen among 7, 9, and 10) into 16

Iteration 4: merge 10 and 16 into 26

Cost = 2*3 + 3*3 + 5*2 + 7*2 + 9*2 = 57.

Job Sequencing with deadlines

- We are given a set of n jobs. Associated with jobs I is an integer deadline di >= 0 & a profit pi>= 0. For any job I the profit pi is earned iff the job is completed by its deadline. Only one machine is available for processing jobs. A feasible solution for this problem is a subset of jobs such that each job in this Subset can be completed by its deadline

Example

- Let n=4
- (P1, P2,P3,P4)= (100,10,15,27)
- (d1, d2,d3,d4)= (2,1,2,1)

(1,2) = 110

(1,3)= 115

(1,4) = 127

(2,3)= 25

(3,4)= 42

1 100

2 10

3 15

4 27