Outline

- Divide and conquer (work trees)
- Heap-sort
- Quick-sort

Last time … Merge-sort

Merge-sort(S={s₁, s₂, ..., sₙ})
  If n=1 return(S)
  Else
    S₁ = Merge-sort(s₁, ..., sₙ/2)
    S₂ = Merge-sort(sₙ/2+1, ..., sₙ)
    Return Merge(S₁, S₂)

Divide and Conquer

- “Divide and conquer” is an algorithmic technique:
  – Break the problems into sub-problems of size n/b
  – Solve the sub-problems
  – Combine the solutions to the sub-problems to create a solution for the original problem
- “Divide and conquer” recurrence relations
  – T(1)=c=f(1)
  – T(n)=a T(n/b) + f(n)

Work Tree for Divide and Conquer

First consider the case n=bᵐ

Total work: c

Work Tree for Divide and Conquer

First consider the case n=bᵐ

A root with a sub-trees
  – Root
    Input Size: n=bᵐ
    Work: f(n)
  – Each child
    Roots a work tree with Input Size bᵐ⁻¹
Work Tree for Divide and Conquer

Properties of nodes at level \( i \) (root is at level 0):
- Input size: \( n/b^i \)
- Work: \( f(n/b^i) \)

Properties of level \( i \):
- Number of nodes at level \( i \): \( a \)
- Total work of nodes at level \( i \): \( af(n/b^i) \)

Property of tree:
- Number of levels: \( m+1 \)
- Total work: \( \sum_{i=0}^{m} af(n/b^i) = ? \)

Where is “most” of the work?
\( f(n) \) is slow-growing \( \leftrightarrow \) \( f(n) \) is fast-growing

\[ f(n) \]
\[ af(n/b) \]
\[ a^2f(n/b^2) \]
\[ a^m=0 \]
\[ n^m/0 \]

\( f(n)=cn, m=\log_b n \)

\[ f(n) = cn \]
\[ af(n/b) = cn(a/b) \]
\[ a^2f(n/b^2) = cn(a/b)^2 \]

\[ T(n) = cn \sum_{i=0}^{m} (a/b)^i \]

Total Work

\[ T(n) = cn \sum_{i=0}^{m} (a/b)^i \]

\( a<b: \) \( \mathcal{O}(n) \)
\( a=b: \) \( \mathcal{O}(n \log_b a) \)
\( a>b: \) \( \mathcal{O}(n^{\log_b a}) \)

\( f(n)=cn^k \)

\[ f(n) = cn^k \]
\[ af(n/b) = cn^k(a/b^k) \]
\[ a^2f(n/b^2) = cn^k(a/b^{2k})^2 \]

\[ a^m=0 \]
\[ n^m/0 \]

\[ T(n) = cn^k \sum_{i=0}^{m} (a/b)^i \]
Total Work

\[ T(n) = cn^k \sum_{i=0}^{m} \left( \frac{a}{b} \right)^i \]

- \( a < b \):
- \( a = b \):
- \( a > b \):

Heaps

A heap is a data-structure for storing integer that supports:

1. Build-heap(S): Return a heap on the integers in the set S.
2. Insert(x,H): Insert the integer x into the heap H.
3. Find-min(H): Return the smallest integer in the heap H.
4. Extract-min(H): Remove the smallest integer from the heap H and return it.

Outline

- Divide and conquer (work trees)
- **Heap-sort**
- Quick-sort
Heap: \{7,1,5,4,2,6\}

1. Rooted, binary tree, filled level by level from the left.
2. Heap property: the integer stored at a node is no larger than those of its descendents.

Heap

A heap is a data-structure for storing integer that supports:

1. Build-heap(S): Return a heap on the integers in the set S.
2. Insert(x,H): Insert the integer x into the heap H.
3. Find-min(H): Return the smallest integer in the heap H.
4. Extract-min(H): Remove the smallest integer from the heap H and return it.

Insert(3,H) – Step 1 (add)

Insert(3,H) – Step 2 (bubble up)

Insert(3,H) – return
Heap

A heap is a data-structure for storing integer that supports:
1. Build-heap(S): Return a heap on the integers in the set S.
   \(O(\log n)\)
2. Insert(x,H): Insert the integer x into the heap H.
   \(O(1)\)
3. Find-min(H): Return the smallest integer in the heap H.
4. Extract-min(H): Remove the smallest integer from the heap H and return it.

Extract-min(H) – Step 1 (remove)

Extract-min(H) – Step 2 (move last to root)

Extract-min(H) – Step 3 (bubble down)

Extract-min(H) – return
Heap

A heap is a data-structure for storing integer that supports:

1. \text{Build-heap}(S): Return a heap on the integers in the set \( S \).
   \( \mathcal{O}(\lg n) \)

2. \text{Insert}(x,H): Insert the integer \( x \) into the heap \( H \).
   \( \mathcal{O}(1) \)

3. \text{Find-min}(H): Return the smallest integer in the heap \( H \).
   \( \mathcal{O}(1) \)

4. \text{Extract-min}(H): Remove the smallest integer from the heap \( H \) and return it.
   \( \mathcal{O}(\lg n) \)

---

Build-heap\{7,1,5,4,2,6\}

Heap

Bubble down

Build-heap\{7,1,5,4,2,6\}

Heap

Bubble down

Build-heap\{7,1,5,4,2,6\}

Heap

Bubble down

Build-heap\{7,1,5,4,2,6\}
Running Time

Level 0: 1 node, takes $O(\lg(n))$
Level 1: 2 nodes, each takes $O(\lg(n/2))$
Level $i$: $2^i$ nodes, each takes $O(\lg(n/2^i))$
Level $\lg(n/2)$: $n/2$ nodes, each takes $O(\lg(n/2^{\lg(n/2)}))$
Time: $\sum_{i=0}^{\lg(n/2)^{\lg(n/2)}} = O(n)$

Implementing a heap in an array

Array Indexing

Heap-sort(S)

Quick-sort(S)

Outline

- Divide and conquer (work trees)
- Heap-sort
- Quick-sort
Quick-sort(3,1,5,2,4)
Pivot rule: Choose first element in list
Quick-sort(3,1,5,2,4)
Quick-sort(1,2), 3, Quick-sort(5,4)
Quick-sort(), 1, Quick-sort(2), 3, Quick-sort(5,4)
1, 2, 3, Quick-sort(5,4)
1,2,3, Quick-sort(4), 5, Quick-sort()
1,2,3,4,5

Analysis
• Quick-sort is correct: Inductive argument
• Quick-sort is $O(n^2)$
• Average case analysis of quick-sort.

Average-case analysis
What does average-case mean?
– Deterministic algorithm with a known input distribution
– Randomized algorithm on any (i.e. worst-case) input

Suppose …
• We have a pivot rule such that for some $d>1$
  – $n/d$ elements are no less than the pivot
  – $n/d$ elements are no greater than the pivot
• $T(n) \leq T(n/d) + T((d-1)n/d) + c$

Work Tree
Input = n
Work = cn

Running Time
• Work done at level $i$: $O(cn)$
• Number of levels: $O(\log_{(d-1)/d} n)$
• Running time: $O(n \log(n))$
Randomized Quick-sort(S)

- If $|S| \leq 1$ then return
- Choose a pivot $s$ uniformly at random from $S$
- $S_1 = \{ t \in S - \{s\} | t > s \}$
- $S_2 = \{ t \in S - \{s\} | t \leq s \}$
- Return Quick-sort($S_1$), $s$, Quick-sort($S_2$)

Average-case analysis

What does average-case mean?
- Deterministic algorithm with a known input distribution
- Randomized algorithm on any (i.e. worst-case) input

A brief tour of (discrete) probability theory…

- Sample space and elementary events
- Discrete probability distributions
- Discrete random variables
- Expectation

The experiment

- A fair coin is flipped
- Sample space: $\{\text{Head}, \text{Tail}\}$

Discrete Probability Distribution

Assigns a real number to outcomes:
- Experiment 1: $P(H) = P(T) = 1/2$
- Experiment 2:
  $P(\text{HH}) = P(\text{HT}) = P(\text{TH}) = P(\text{TT}) = 1/4$
Discrete Probability Distribution Properties

- \( P(A) \geq 0 \)
- \( P(S) = 1 \)
- \( P(A \cup B) = P(A) + P(B) \) when \( A \) and \( B \) are disjoint events.

Discrete Random Variable \( X \)

- Sample space is a finite set of real numbers.
- \( X \) is the outcome of the experiment.
- Probability distribution: \( P(X=x) \)
- Expected value of \( X \):
  \[
  E[X] = \sum_{x \in S} x \cdot P(X=x)
  \]
  \[
  E[X^2] = \sum_{x \in S} x^2 \cdot P(X=x)
  \]
  \[
  \text{Var}(X) = E[(X-E[X])^2]
  \]

Example

- A unfair coin is tossed \( n \) times:
  \( P(H)=p, P(T)=1-p=q \)
- \( X \) is the number of heads
- \( P(X=k) = C(n, k) \cdot p^k \cdot q^{n-k} \)
  (Binomial distribution)
- \( E[X] = np \)
- \( E[X^2] = npq + n^2p^2 \)