
2 SORTING BY RANKING

{**rank** of an item=number of items smaller}

rank each item by counting;

then place each item according to its rank.

If duplicate, then place it at the nearest slot to the right.

$$W(n) = O(n(n-1)) = A(n) = B(n)$$

$$S(n) = O(n)$$

To handle duplicates, redefine

$\text{rank}(i)$ = number of items smaller than i or, if equal, occurring before i .

Oblivious Algorithm

1 _____

3 SORTING BY SWAPPING

Bubble Sort—good when input is nearly sorted

$$W(n) = O(n^2/2)$$

$$S(n) = O(1)$$

Odd-Even Exchange Sort

a) odd-even compare & exchange

b) even-odd compare & exchange

c) repeat step (a) and (b) if exchange-count > 0

$$W(n) = O(n^2)$$

$$S(n) = O(1)$$

4 SORTING BY INSERTION

Online algorithms.

Linear Insertion Sort

Insert the next element in the ordered list prepared so far by sequential search
& shifting.

$$W(n) = O(n^2/2)$$

$$S(n) = O(1)$$

$O(n)$ time on a sorted list

Binary Insertion sort

perform binary search to find location for insertion.

$$W(n) = O(n \log n) + O(n^2).$$

Tree Sort

Insert into a binary search tree, then traverse tree in-order.

$$W(n) = O(n^2)$$

$$S(n) = O(n)$$

$$A(n) = B(n) = O(n \log n) + O(n)$$

5 SORTING BY SELECTION

Offline algorithms

Selection sort

find maximum & replace with the concurrent last

$$W(n) = O(n^2)$$

$$S(n) = O(1)$$

$O(n^2)$ even on a sorted list

Tournament Sort

$$W(n) = A(n) = B(n) = O(n \log n)$$

$$S(n) = O(n) \text{ for the tournament tree}$$

Heap Sort

- construct a max heap $-O(n)$
- delete root and update heap repeatedly $-O(n \log n)$

$$W(n) = A(n) = O(n \log n)$$

$$S(n) = O(1)$$

5.1 Heap Sort

- Restore-Heap(i): $O(h)$ where h is the height of the node i .

- Construct Heap:

For $i = \lfloor \frac{n}{2} \rfloor$ down to 1 Restore-Heap(i)

$O(n)$

- Heap Sort:

1. Construct Heap $- O(n)$

2. For $i:=n$ down to 2 $- O(n \log n)$

exchange $L[1]$ with $L[i]$

decrement heap size

Restore-Heap(1)

- Time Complexity of deletion phase in Heap Sort

$$W(n) = 2 \log n + W(n-1)$$

$$= 2 \sum_{i=1}^n \log i$$

$$\leq 2 \int_1^{n+1} \log x dx$$

$$= [2(x \log x - x)]_1^{n+1}$$

$$= 2n \log n - 2n$$

5.1.1 Heapsort Construction

Construct Heap:

for $i = \lfloor n/2 \rfloor$ downto 1

 Restore-heap(i)

Time Complexity of Iterative Algorithm for Heap Construction:

$$\sum_{h=1}^{\lfloor \log n \rfloor} \lceil n/2^{h+1} \rceil O(h)$$

$$= O\left(n \sum_{h=1}^{\log n} (h/2^h)\right)$$

$$= O(n) \text{ (pp. 159)}$$

Consider $\sum_{h=1}^{\log n} h/2^h$

Let

$$x = 1/2 + 2/2^2 + 3/2^3 + 4/2^4 + \cdots + y/2^y \quad (1)$$

$$2x = 1 + 2/2^1 + 3/2^2 + 4/2^3 + \cdots + y/2^{y-1} \quad (2)$$

$$x = 1 + 1/2 + 1/2^2 + \cdots + 1/2^{y-1} \quad (3)$$

$$-y/2^y \quad (4)$$

$$= \frac{(1/2)^y - 1}{1/2 - 1} - y/2^y \quad (5)$$

$$= 2(1 - (1/2)^y) - y/2^y \quad (6)$$

$$\leq 2 \quad (7)$$

5.1.2 Heapsort: Recursive Construction

Construct-Heap(n):

construct left subheap

construct right subheap

Restore-heap(1)

Time Complexity:

$$W(n) = 2w(n/2) + \log n \quad (9)$$

$$= \log n + 2w(n/2) \quad (10)$$

$$= \log n + 2 \left(\log(n/2) + 2w\left(\frac{n/2}{2}\right) \right) \quad (11)$$

$$= \log n + 2 \log n - 2 \log 2 + \quad (12)$$

$$2^2 w(n/2^2) \quad (13)$$

$$= \log n + 2 \log n - 2 \log 2 + \quad (14)$$

$$2^2 \left(\log\left(\frac{n}{2^2}\right) + 2w\left(\frac{n/2^2}{2}\right) \right) \quad (15)$$

$$= \log n + 2 \log n - 2 \log 2 + \quad (16)$$

$$+ 2^2 \log n - 2^2 \log(2^2) + 2^3 w(n/2^3) \quad (17)$$

$$\dots \quad (18)$$

$$= \log n + 2 \log n - 2 \log 2 + \quad (19)$$

$$+ 2^2 \log n - 2^2 \log(2^2) + \dots + \quad (20)$$

$$2^k \log n - 2^k \log(2^k) + 2^{k+1} w(n/2^{k+1}) \quad (21)$$

$$= \log n (1 + 2 + 2^2 + \dots + 2^k) \quad (22)$$

$$- (2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \dots + k \cdot 2^k) \quad (23)$$

Here, we assume that $n = 2^k$ so $k = \log n$.

$$\begin{aligned} \text{Let } s &= 2^0 + 2^1 + 2^2 + \cdots + 2^k \\ &= \frac{2^{k+1}-1}{2-1} = 2^{k+1} - 1 \end{aligned}$$

To derive this formula, one can follow these steps:

$$s = 2^0 + 2^1 + 2^2 + \cdots + 2^k \quad (24)$$

$$2s = 2^1 + 2^2 + \cdots + 2^k + 2^{k+1} \quad (25)$$

$$s = -1 + 2^{k+1} \quad (26)$$

Thus, $\sum_{i=0}^k 2^i = 2^{k+1} - 1$

Likewise, let

$$T = 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + k \cdot 2^k \quad (27)$$

$$2T = 1 \cdot 2^2 + 2 \cdot 2^3 + \cdots + \quad (28)$$

$$(k-1) \cdot 2^k + k2^{k+1} \quad (29)$$

$$T = -2^1 - 2^2 - 2^3 - \cdots - 2^k + \quad (30)$$

$$k2^{k+1} \quad (31)$$

$$= k2^{k+1} - (2^1 + 2^2 + \cdots + 2^k) \quad (32)$$

$$= k2^{k+1} - (2^{k+1} - 2) \quad (33)$$

$$= (k-1)2^{k+1} + 2 \quad (34)$$

$$(35)$$

$$\begin{aligned} \Rightarrow W(n) &= \log n(2^{K+1} - 1) - ((k-1)2^{k+1} + 2) \\ &= 2n \log n - \log n - 2n \log n + 2n - 2 \\ &= 2n - \log n - 2 \\ &= O(n) \end{aligned}$$

5.1.3 Priority Queue employing Heap

Read pp. 162 (Section 6.5)

Delete Max/Min

Insert

See Exercise 6-1, pp. 166

6 SORTING BY MERGING

- **Mergesort**

- $$\begin{aligned} W(n) &= O(n) + 2w(n/2) \\ &= n + 2w(n/2) \\ &= n + 2(n/2 + 2w(n/2^2)) \\ &= n + n + 2^2w(n/2^2) \\ &= 2n + 2^2w(n/2^2) \\ &= 2n + 2^2(n/2^2 + 2w(n/2^3)) \\ &= 3n + 2^3w(n/2^3) \\ &\dots \\ &= kn + 2^kw(n/2^k) \\ w(n/2^k) &= w(1) = 0 \\ n/2^k &= 1, k = \log_2 n \\ \Rightarrow W(n) &= \theta(n \log n) \end{aligned}$$

- Recursion tree for $W(n)$

- $S(n) = O(n)$

Can be reduced to $O(1)$ but algorithm slows down.

A temporary array of half the size is required, however.

7 SORTING BY SPLITTING

$W(n) = O(n^2)$ but $A(n) = O(n \log n)$

Quicksort (A, p, r) :

if $p < r$ then $q \leftarrow \text{Partition}(A, p, r)$

 Quicksort (A, p, q)

 Quicksort $(A, q + 1, r)$

7.1 Quicksort: Partitioning

Partition I.

$x := A[p]; i := p - 1; j := r + 1$

while (TRUE) **do**

Repeat Decrement j **until** $A[j] \leq x$

Repeat Increment i **until** $A[i] \geq x$

if $i < j$

then exchange $A[i]$ and $A[j]$

else return j

endwhile

$$x = 15$$

15 7 23 5 20 3

Why do i and j never get out of array bounds?

$W(n) = n + 2$ comparisons

$$W(n) = O(n^2) = O(n) + W(n-1)$$

$$B(n) = O(n) + 2B(n/2) = O(n \log n)$$

Balance Partitioning:

- Even if each split is 1% on one side and 99% on the other, recursion tree remains logarithmic.

- Alternate good and bad splits:

Quicksort Improvements

- random x
- median of first, middle and last items
- insertion sort for $n \leq 15$

$$S(n) = O(n)$$

(See 7-4; reduces $S(n)$ to $O(\log n)$)

7.2 Quicksort: Partitioning II

Input: $A[p..r]$

Invariants: $\{\text{All items in } A[2..i] \text{ are } < \text{the pivot.}\}$

$\{\text{All items in } A[(i+1)..(unknown-1)] \text{ are } \geq \text{the pivot}\}$

$x = A[p]; i := p;$

for $unknown := p + 1$ **to** r **do**

if $A[unknown] < x$ **then**

$i := i + 1; \text{swap}(A[i], A[unknown])$

$\text{swap}(A[1], A[i])$

$W(n) = n - 1$ comparisons

Question 1: State the tree (insertion) sort, odd-even transposition sort, and quicksort algorithms in 3-4 sentences each, and estimate, with suitable but brief justification, their (i) average case time complexities and (iii) their worst-case space complexities. Use known properties of binary search trees in case of tree sort and intuitive arguments about the other two to justify average case scenarios.

Question 2: State a situation with suitable but brief justification, if any, when you would prefer (i) linear insertion sort over binary insertion sort, (ii) heapsort over quicksort, (iii) quicksort over merge sort, (iv) tournament sort over sorting by ranking, and (v) odd-even exchange sort over bubble sort.

Question 3: Set up a recurrence relation for the depth of the stack for quicksort assuming that the sizes of the left and the right partitions are compared, and quicksort is recursively called on the smaller partition only. There is no recursive call on the larger partition because of tail recursion. By solving this recurrence relation, show that the depth of the stack is bounded by $O(\log n)$ on a list of n integers.

Question 4: Prove that a lower bound on problem complexity of comparison-based sorting algorithms is $(n \log n)$.

Question 5: Workout Quicksort by employing partition-2 algorithm (where in two pointers both move left to right). Show one snapshot of the array at each level of recursion after performing partitioning of all subarrays using first item as pivot. 3 5 1 9 4 2.

Question 6: State sorting by ranking, binary insertion sort, and heapsort algorithms in 3-4 sentences. Then, set up the time complexity function of jump search with jump size of k and prove that best choice for jump size k is \sqrt{n} .

Question 7: Set up a recurrence relation for the worst-case time complexity of merge sort, simplify it by dropping insignificant constants, and (a) obtain a closed form solution using unfolding the recursion, finding a pattern, and analytically solving it, and (b) using estimation from its recursion tree.

Question 8:

a) Briefly state how odd-even transition sort can be parallelized to obtain best speedup, and justify its fastest parallel execution time in the worst-case using n processors (T_n) on an input of length n using big-oh notation

(b) Repeat the above for merge sort, recalling that binary search can be employed.

7.3 Av. Case Complexity of Quicksort

Assume

- all keys distinct
- all permutations equally likely

Probability that split point is i , for $1 \leq i \leq n$, is $1/n$

$$\begin{aligned} A(n) &= (n-1) + \frac{1}{n}(A(0) + A(n-1) + \\ &A(1) + A(n-2) + \cdots + A(n-1) + A(0)) \\ &= n-1 + \frac{2}{n}(A(0) + A(1) + \cdots + A(n-1)) \\ A(n) &= n-1 + \frac{2}{n}\sum_{i=2}^{n-1}A(i), \quad A(0) = A(1) = 0 \\ (n)A(n) &= (n)(n-1) + 2\sum_{i=2}^{n-1}A(i) \\ A(n-1) &= n-2 + \frac{2}{n-1}\sum_{i=2}^{n-2}A(i) \\ (n-1)A(n-1) &= (n-1)(n-2) + 2\sum_{i=2}^{n-2}A(i) \\ nA(n) - (n-1)A(n-1) & \\ &= n(n-1) - (n-2)(n-1) + 2A(n-1) \end{aligned}$$

$$nA(n) - (n+1)A(n-1) = 2(n-1)$$

$$\frac{A(n)}{n+1} - \frac{A(n-1)}{n} = \frac{2(n-1)}{n(n+1)}$$

Let $B(n) = \frac{A(n)}{n+1}$ (*Changing Variable*)

Since $A(1) = 0$, $B(1) = 0$

$$\Rightarrow B(n) - B(n-1) = \frac{2(n-1)}{n(n+1)}$$

$$B(n) = \frac{2(n-1)}{n(n+1)} + B(n-1)$$

$$= \frac{2(n-1)}{n(n+1)} + \left(\frac{2(n-2)}{(n-1)n} + B(n-2) \right)$$

$$= B(1) + \frac{2 \cdot 1}{2 \cdot 3} + \frac{2 \cdot 2}{3 \cdot 4} + \cdots + \frac{2(n-1)}{n(n+1)}, B(1) = 0$$

$$B(n) = \sum_{i=2}^n \frac{2(i-1)}{i(i+1)}$$

$f(n) = \frac{2(n-1)}{n(n+1)}$ continuous decreasing function Sum_a^b f(x) <= Int_{a-1}^b f(x)dx - pp. 51

$$B(n) \leq \int_2^n f(x)dx, \quad f(1) = 0$$

$$\frac{2(x-1)}{x(x+1)} = \frac{A}{x} + \frac{B}{x+1}$$

$$= \frac{(A+B)x + A}{x(x+1)}$$

$$\Rightarrow A = -2$$

$$A + B = 2$$

$$\Rightarrow B = 4$$

$$\begin{aligned}
&\Rightarrow f(x) = \frac{4}{x+1} - \frac{2}{x} \\
&\int_2^n f(x)dx = \int_2^n \left(\frac{4}{x+1} - \frac{2}{x} \right) dx \\
&= (4 \ln(x+1) - 2 \ln x) \Big|_2^n \\
&= 4 \ln(n+1) - 2 \ln n - 4 \ln 3 + 2 \ln 2 \\
&\approx 2 \ln n \\
&\Rightarrow B(n) \leq 2 \ln n \\
&A(n) = (n+1)B(n) \leq 2(n+1) \ln n \\
&\Rightarrow A(n) \leq 1.4(n+1) \log_2 n
\end{aligned}$$

8 LOWER BOUND

(a) Local exchange only

each accomplishes in undoing one inversion

5 1 4 7 2 has inversions (5,1),(5,4),(5,2),(4,2),(7,2)

(n n-1 ... 2 1) has $n(n-1)/2$ inversions

$\Rightarrow O(n^2/2)$ lower bound

(b) lower bound on comparison based sorting

example: a b c - there are 3! outputs.

(for n numbers there are n! outputs)

Every decision tree to sort n numbers must have atleast n! leaf nodes

Every decision tree to sort n numbers must have atleast $2n! - 1$ nodes

Every decision tree to sort n numbers must have depth atleast $\log_2 n!$ leaf nodes

Depth is the lower bound worst case time complexity for this class of algorithms

$$\log_2 n! = \log_2(n * (n-1) * (n-2) * (n-3).....1)$$

$$\log_2 n! = \log_2(n) + \log_2(n-1) + \log_2(n-2) + \log_2(n-3)..... + \log_2(1)$$

$$\log_2 n! = \sum_{j=1}^n \log_2 j$$

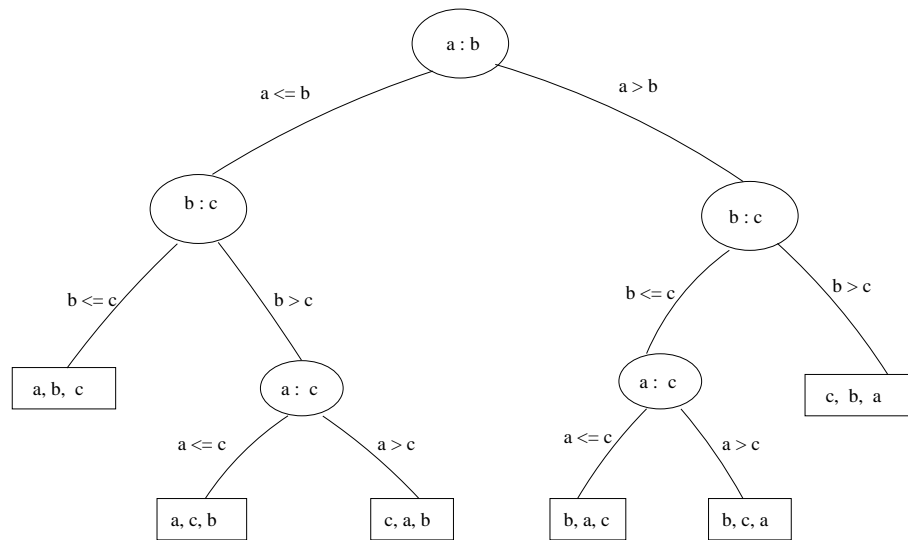
$$\leq \int_1^n \log_2 x dx$$

$$= (x \log x - x)|_1^n$$

$$= n \log n - n$$

Lower Bound on Comparison-Based Sorting Algorithm

Decision Tree for 3 numbers



Depth of binary tree with $n!$ leaves

$$\geq \log_2 n!$$

$$\geq \int_1^n \log x dx$$

$$\geq n \log n - n$$

9 SHELL SORT (Donald Shell)

Sort subarray comprising every h_i location, for a few selected hop sizes h_i ,
 $k \geq i \geq 1$, and final $h_1 = 1$
 $h_1 = 1$ always use insertion sort for sorting

with $h_2 = 1.72n^{1/3}$

$$\begin{aligned} W(n) &= \left(\frac{n}{1.72n^{1/3}}\right)^2 + 1.72n^{1/3} + n^2 \\ &= \frac{n^2}{1.72n^{1/3}} + n^2 \\ &= \frac{n^{5/3}}{1.72} + n^2 \\ &= O(n^2) \end{aligned}$$

for $h_k = 2^k - 1$, $1 \leq k \leq \lfloor \log n \rfloor$
 $W(n) = O(n)$

$$\begin{aligned} 2^5 - 1 &= 31 = (11111)_2 \\ 2^4 - 1 &= 15 = (1111)_2 \end{aligned}$$

for h_k is an integer of the form $2^i 3^j$, $h_k < n$
 $W(n) = O(n(\log n)^2)$

$$\begin{array}{c} 2^0 3^0, 2^1 3^0, 2^0 3^1, 2^1 3^1, 2^2 3^1 \\ 1 \ 2 \ 3 \ 6 \ 12 \end{array}$$

too many h'_k s, hence overhead is large.