CS F364: Design & Analysis of Algorithm

03 R Quick Sort
Defective Chessboard

Dr. Kamlesh Tiwari
Assistant Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Jan 22, 2021 ONLINE (Campus @ BITS-Pilani Jan-May 2021)

http://ktiwari.in/algo

Quick Sort

Which algorithm is better ?

	Best Case	Worst Case	Average Case
Algo-01	$n \log n$	$n \log n$	$n \log n$
Algo-02	$n \log n$	$n(n-1)$	$n \log n$

- If I tell you Algo-01 is merge sort and Algo-02 is quick sort then?
- Quick sort is popular because it always behaves like average case as the input size increases

Table: 1000 execution of randomized quick sort on random list

Number of times runtime exceed the average behavior	Input size (# of items) 10^2	10^3	10^4	10^5	10^6
10%	190	49	22	10	3
20%	28	17	12	3	0
50%	2	1	1	0	0
100%	0	0	0	0	0

Randomized Quick Sort

```
QuickSort(A,p,r)
  if p<r
    q = Partition(A,p,r)
    QuickSort(A,p,q−1)
    QuickSort(A,q+1,r)
```

```
Partition(A,p,r)
  x = A[r]
  i = p−1
  for j = p to r−1
    if A[j] <= x
       i = i + 1
       swap A[i] <−> A[j]
  swap A[i+1] <−> A[r]
  return i+1
```

```
rQuickSort(A,p,r)
   if(p<r)
      q = rPartition(A,p,r)
      rQuickSort(A,p,q−1)
      rQuickSort(A,q+1,r)
```

```
rPartition(A,p,r)
  i = random(p,r)
  swap A[r] <−> A[i]
  return Partition(A,p,r)
```


Analysis of Randomized Quick Sort

Estimate number of comparisons performed during execution

- Let sorted list $< S_1, S_2, S_3, ..., S_n >$ with S_i as i^{th} smallest element
- Define random variable X_{ij} as number of comparisons between S_i and S_j. X_{ij} could take a value 0 or 1
- Expected number of comparison is

$$E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$$

- If p_{ij} be the probability of comparison between S_i and S_j. Then,

$$E[X_{ij}] = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij}$$

Randomized Quick Sort

21	15	14	18	31	6	17	11	15	8	19	16
0	1	2	3	4	5	6	7	8	9	10	11

6	8	11	14	15	15	16	17	18	19	21	31
0	1	2	3	4	5	6	7	8	9	10	11
		↑i					↑j				

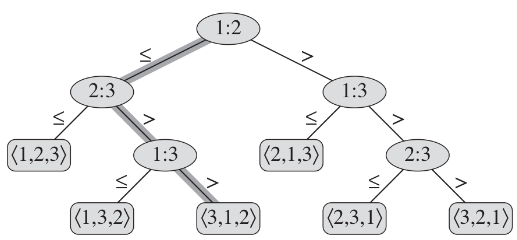
- Pivot element can either be
  1. S_i or S_j: its probability is $\frac{2}{j-i+1}$
  2. Inside S_q from $i < q < j$, comparison not possible
  3. Outside S_r from $r < i$ or $j < r$, no effect on comparison

$$\sum_{i=1}^{n}\sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} p_{ij} = \sum_{i=1}^{n}\sum_{j>i} \frac{2}{j-i+1} = 2\sum_{i=1}^{n}\sum_{k=1}^{n-i+1}\frac{1}{k}$$
$$\leq 2\sum_{i=1}^{n}\sum_{k=1}^{n}\frac{1}{k} = 2nH_n = O(n \ln n)$$

as $H_n \sim \ln n + \Theta(1)$

Decision Tree Model of Sorting

Sort three items a_1, a_2, a_3



Is it always to be a binary tree?
What is worst case time taken by this algorithm? $O(height)$
How many leaves would be there with 4 items?

Lower Bound of Sorting

Any comparison sort needs $\Omega(n \log n)$ comparisons in the worst case.

- There are $n!$ permutations of n items. Each should be at leaf
- Binary tree of height h has at most 2^h leaves

$$n! \leq 2^h$$
$$h \geq \log(n!)$$

- Stirling's approximation of $n!$ is $(n/e)^n$

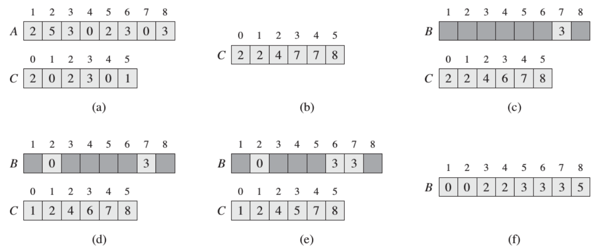
$$h \geq n\log(n) - n\log(e)$$
$$h = \Omega(n\log(n))$$

Wish to sort in linear time $O(n)$? use Counting Sort

```
CountingSort(A,B,k)
  Let C[0..k] be new array of zeros

  for j=1 to A.length
    C[A[j]] = C[A[j]] + 1

  for i = 1 to k
    C[i] = C[i] + C[i-1]

  for j = A.length down to 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] -1
```

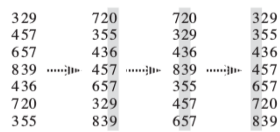
Apply to Sort: 2, 5, 3, 0, 2, 3, 0, 3

Counting Sort in action



Radix Sort

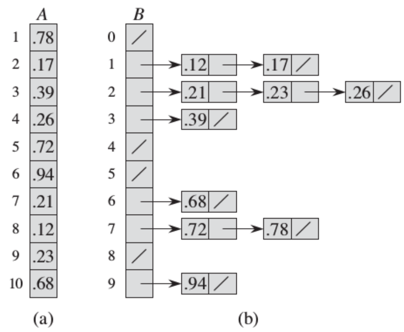
Use a stable sorting algorithm to sort array A on digit (1 to d)



Bucket Sort

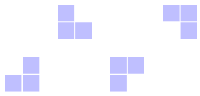
```
bucketSort(A)

  n = A.length
  Let B[0..n-1] be new array

  for i = 0 to n-1
    make B[i] as empty list

  for i = 1 to n
    inset A[i] into list B[[nA[i]]]

  for i = 0 to n-1
    Sort list B[i] with insertion sort

  concatinate B[0], B[1], ..., B[n-1] in order
```


Bucket Sort



Defective Chessboard

- Consider a chessboard of size $2^k \times 2^k$ where one cell is defective. Your task to cover it using a triomino.



Obviously

- Triomino cannot cover the defected one
- Triomino should not overlap
- Triomino must cover all other squares

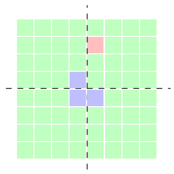
Note: $4^k - 1$ is divisible by 3

Defective Chessboard

Divide and conquer

1. **Divide:** in smaller size instances.
2. **Recursive step:** use same framework till it is trivially solvable
3. **Conquer:** combine solutions of smaller instances to get overall solution

- $2^k \times 2^k$ size board is divided in four $2^{k-1} \times 2^{k-1}$ size board
  - Let $T(n)$ be time to tile $2^k \times 2^k$ board



$$\begin{aligned} T(n) &= t_d + 4T(n/2) + t_c = 4T(n/2) + c \\ &= c + 4c + \ldots + 4^{n-2}c + 4^{n-1}d \\ &= c\frac{4^{n-1} - 1}{4 - 1} + 4^{n-1}d \\ &= \left(\frac{c}{12} + \frac{d}{4}\right) \times 4^n - \frac{c}{3} = \Theta(4^n) \end{aligned}$$

Thank You!

Thank you very much for your attention! (Reference[1])

Queries ?

[1] [1] Book - *Introduction to Algorithm*, By THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST, CLIFFORD STEIN