CS/IS C363 Data Structures & Algorithms

Review: Efficiency & Complexity
Sorting

Time and Space Complexity

- Order Complexity and Notation
- Examples

Cost Models

Comparative Review:

Sorting by Insertion and Merging

Analysis

QuickSort

- A Divide-and-Conquer Algorithm
- Best and Worst cases
  - Analysis

Linear Search Algorithm

```
// A indexed from 1 to N

index= 1;

while (index <= N) {

    if (A[index] = = x)

            return x;

}

return Not_Found;
```

Worst case:

- Loop executes until index == N i.e. until the end of list is reached
- Size halved in each iteration   N, N-1, N-2, … 1
-

Time complexity O(N)

Best Case ?
Average Case ?

Binary Search Algorithm

```
// A indexed from 1 to N

low = 1;  high = N;

while (low <= high) {

     mid = (low + high) /2;

    if (A[mid] = = x)  return x;

    else if (A[mid] < x) low = mid +1;

     else high = mid – 1;

}

return Not_Found;
```

Worst case:

- Loop executes until low > high i.e. until size of list becomes 0
- Size halved in each iteration N, N/2, N/4, … 1
- Number of steps is K such that 2K = N i.e. logN steps where N is input size

Time complexity O(logN)

Time Complexity

- Polynomial Time Complexity
  - Time Complexity is **O(Nk)** for some constant k, where N is input size.
- Exponential Time Complexity
  - Time Complexity is **O(2N)**, where N is the input size.

Time Complexity

- Consider the following algorithm:

```
int fact(int N)   {
      j=1;  prod=1;
      while (j<=N) {
           j=j+1; prod=prod*j;
       }
      return prod;
}
```

What is the time complexity?

Is this polynomial time? Why or why not?

- Uniform Cost vs. Logarithmic Cost
  - Uniform Cost – All basic operations cost same (constant) amount of time (irrespective of the data size)
  - Logarithmic Cost – Each operation has a cost that is proportional to the size of the data
  - Hint:
    - Refer to the RAM model slide for assumptions;
    - consider the call *fact(100).*

Sorting

Comparative Review:

Sorting by Insertion and Merging

Analysis

QuickSort

- A Divide-and-Conquer Algorithm
- Best and Worst cases
  - Analysis

Complexity of Sorting Algorithms

- Insertion Sorting:
  - Time taken for sorting N elements:
  - Time taken for sorting N-1 elements +
  - Time taken for inserting 1 element into sorted list
  - Time taken for inserting 1 element into sorted list:
  - For finding the position (say the element is in Kth position): K
  - For shifting the rest of the elements: N-K
  - Total time taken for insertion in sorted list is $\Theta(N)$
  - Finding by binary search – does it improve time complexity?

Note on Notation

- Notation: (lower bound – asymptotic lower bound)

Ω(f(N)) = { g(n) | there exists +ve consts. c1 and n0 s.t.

c1*f(N) <= g(n) for all n>n0 }

- Notation: (tight bound – asymptotic upper and lower bound)

ϴ(f(N)) = { g(n) | there exist +ve consts. c1, c2, and n s.t.

c1*f(N) <= g(n) <= c2*f(n) for all n>n0 }

Complexity of Sorting Algorithms

- Insertion Sorting – Worst case (recurrence relation):
  - $T(N) = T(N-1) + \Theta(N)$ for $N>1$
  - $= \Theta(1)$ for $N=1$
- Solution (techniques)
  - By substitution / iteration
- Time complexity:
  - $\Theta(N*N)$
- What about the average case?

Complexity of Sorting Algorithms

- Merge Sorting – Worst case (recurrence relation):
  - T(N) = 2*T(N/2) + Θ(N) for N>1
  - = Θ(1) for N=1
- Solution (techniques)
  - By substitution / iteration
  - T(N) = Θ(N*logN)
  - By Master Theorem.
  - Reading Exercise: Master Theorem
- What about the average case?

Sorting – Comparative Review

- Insertion Sort
  - Divide & Combine: sublist of size N-1, insert
  - Time Complexity:
  - Worst case: O(N*N)
  - Average case: O(N*N)
  - Space Complexity:
  - Worst case: O(1)
  - Insertion Sort is online
  - Insertion requires Random Access

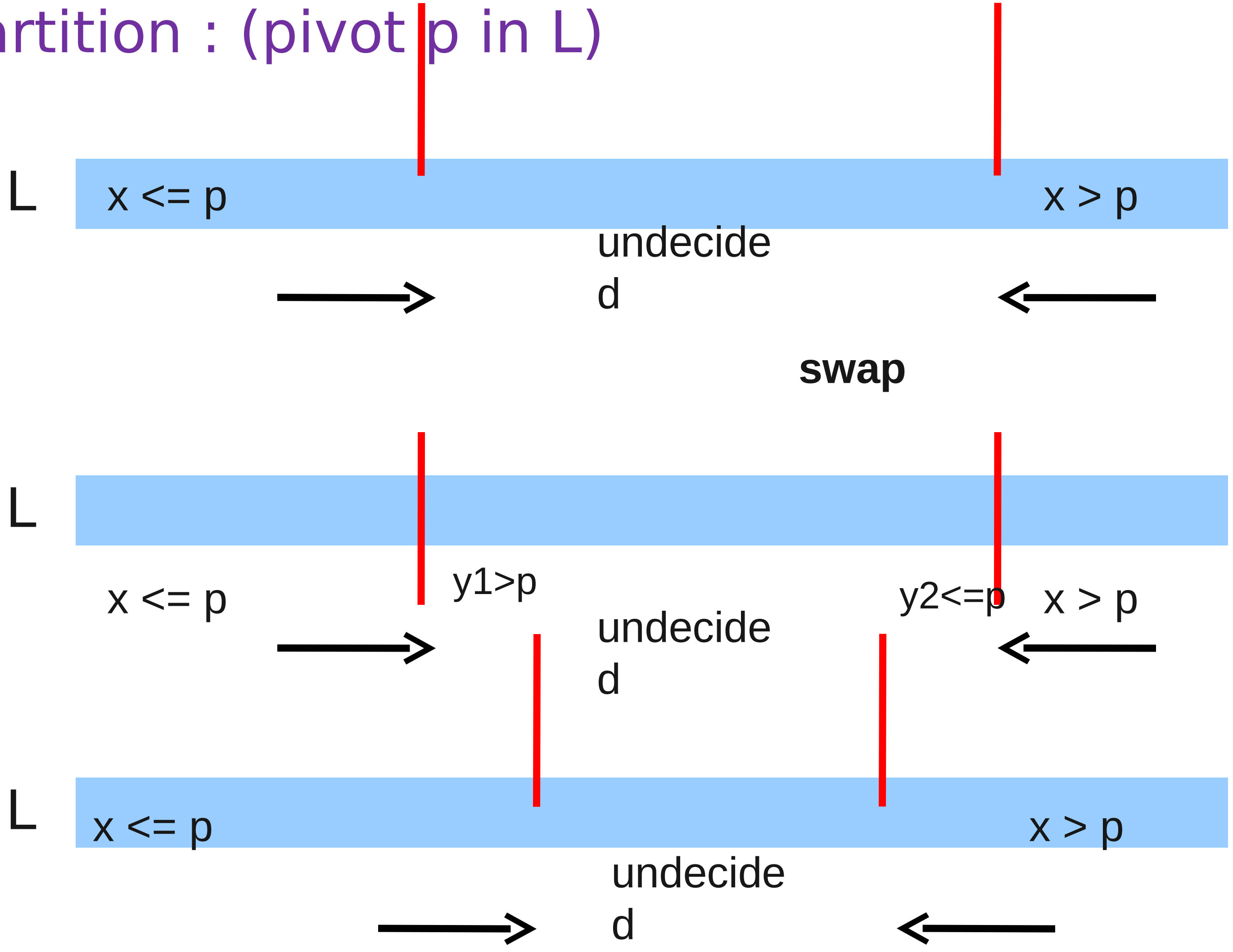
- Merge Sort
  - Divide & Combine: sublists of size N/2, merge
  - Time Complexity:
  - Worst case: O(N*logN)
  - Average case: O(N*logN)
  - Space Complexity:
  - Worst case: O(N)
  - Average case: O(N)
  - Merge Sort is not (fully) online – Why?
  - Merge does not require Random Access

QuickSort

- Input: List of **N** elements **L[0], L[1], … L[N-1]**
- Divide and Conquer:
  - Partition **L** into **LL** and **LG** based on a pivot p in **L** s.t. :
  - **LL = { x in L | x <= p }**
  - **LG = { x in L | x > p }**
  - Combine sorted versions of **LL** and **LG** and **{p}**:
  - Append: **LL, { p }, LG**
    - Append is trivial if **LL** & **LG** were sorted in place and p is in position
    - Sorting in place requires partitioning in place:
      - Question: How do you keep **p** in between?

QuickSort

- Partition : (pivot p in L)



QuickSort

- Input: List of N elements L[0], L[1], … L[N-1]
- Algorithm Schema (Divide-and-Conquer):
  1. Pick a pivot index j. Let p = L[j]
  2. Swap p out (if needed)
  3. Partition L based on p, into LL and LG
  4. Put p in place (at the right position)
  5. Sort LL in place
  6. Sort LG in place

Complexity of Sorting Algorithms

- Complexity of Partition:
  - $\Theta(N)$
- Quick Sort – Time - Worst case (recurrence relation):
  - $T(N) = T(N-1) + \Theta(N)$ for $N>1$
  - $= \Theta(1)$ for $N=1$
  - $T(N) = ?$
- Quick Sort – Time - Best case (recurrence relation):
  - $T(N) = 2*T(N/2) + \Theta(N)$ for $N>1$
  - $= \Theta(1)$ for $N=1$
  - $T(N) = ?$

Case Analysis

- When does the worst case occur?
- When does the best case occur?
- What is the average case complexity?

Sorting - QuickSort

QuickSort

- **- Time Complexity**
- **- Best and worst cases**
  - **- Analyses**
- **- Pivot Selection – Median of 3, Random, Median of Medians**
- **- Special Cases: Small Lists, Equal-Valued Keys**

QuickSort – Time Complexity

- Time Complexity of Partition:
  - $\Theta(N)$ - Why?
- Quick Sort – Time - Worst case (recurrence relation):
  - $T(N) = T(N-1) + \Theta(N)$ for $N>1$
  - $= \Theta(1)$ for $N=1$
  - $T(N) = ?$
  - When does the worst case occur?
- Quick Sort – Time - Best case (recurrence relation):
  - $T(N) = 2*T(N/2) + \Theta(N)$ for $N>1$
  - $= \Theta(1)$ for $N=1$
  - $T(N) = ?$
  - When does the best case occur?

QuickSort – Time Complexity

- Average Case:
  - By assuming input distributions to be random one can compute the average case complexity.
  - Verify that O(N*logN) is a solution to recurrence relation:
  - $T(N) = \Theta(N) + (1/N) * (\sum_{k=1}^{N} (T(k-1) + T(N-k)))$
  - for N>1
  - T(N) = 1 for N <= 1
- But the time taken for a specific input depends heavily on the pivot(s) chosen for partitioning.

Case Analysis

- Best case Partitioning:

N/2				N/2			
N/4		N/4		N/4		N/4	
N/8	N/8	N/8	N/8	N/8	N/8	N/8	N/8

...

- logN steps
- Partitioning work in each step is O(N)

- Time Comlexity: O(NlogN)

Case Analysis

- Worst case Partitioning:

1			N-1
1	1		N-2
1	1	1	N-3

- ...
  - N-1 steps
  - Partitioning work in each step is O(N)
  - Time Comlexity: O(N*N)

Balanced Partitioning

- Better time complexity with balanced partitioning
- Consider the following example scenario:

.1N	0.9N

. 09N	0.09 N	0.81N

…

- Time Complexity:

$T(N) = T(N/10) + T(9N/10) + \Theta(N)$

$<= 2*T(9N/10) + \Theta(N)$

$= O(N*\log_{10/9} (N))$

Pivot Selection

- Biased pivots result in unbalanced partition
  - Pivot p such that p < x for most x in L
  - Symmetrically, p > x for most x in L
- Static/Fixed techniques for pivot selection repeat the bias at every level
  - E.g. First element of the list as a pivot in a (mostly) sorted list
- De-biasing Solution:
  - Adaptive selection of pivots
  - Typically done by sampling the input

Pivot Selection Techniques

- Median of 3:
- Median of Medians:
- QuickSelect (selecting Kth smallest element)

Pivot Selection Techniques

- Median of 3:
  - Median of first, middle, and last element in the (sub)list
  - Exclude these elements from partitioning process
- Median of Medians:
  - For every 5 contiguous elements find the median by direct comparison ==> N/5 medians
  - Obtain the median of these N/5 medians
  - How? Sort? QuickSelect?
- QuickSelect (selecting Kth smallest element)

Pivot Selection - QuickSelect

- Median of Medians:
  - For every 5 contiguous elements find the median by direct comparison ==> N/5 medians
  - Obtain the median of these N/5 medians
  - How? Sort? QuickSelect?
- QuickSelect (selecting Kth smallest element)
  1. Partition the list by a pivot p;
  2. p is in its correct position, say J
  3. if K = J done,

     if K<J select Kth smallest from left sub list

     if K>J select K-J th smallest element from right sub list

Q: What is the time complexity of QuickSelect?

Pivot Selection

- Randomized QuickSort
  - Select pivot index uniformly randomly between first and last (indices).
  - Need a (good) random number generator
- Is there a random number?
  - Random sources
  - Bit Selection
  - Repeated coin toss
  - Cost of random number generation
  - Pseudo-random number generators

Small Lists

- Insertion Sort performs better than QuickSort on small lists.
  - Why?
  - So, what?
- Combine the two!
  - Invoke Insertion Sort inside QuickSort when size of the list is small.
  - Alternatively, ignore, small sized lists inside QuickSort, and do an insertion Sort (on the full list)
  - Question: Why does this work (efficiently)?
  - Time Complexity (expected):
  - O(k*k*N + N*log(N)) where k is the threshold (below which InsertionSort performs better than QuickSort)

Equal Values

- QuickSort performs badly when the same key occurs multiple times
- Solution: 3-way partition
  - Maintain an additional partition for elements equal to the pivot
  - Exercise: Implement this!