CSCI 2132
Software Development

Lecture 18:

Implementation of Recursive Algorithms

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- Function definitions
- Function declarations or prototypes
- Arguments and parameters
- Arguments passed by value
- Call stack
- Simple recursion example: power
- Mergesort algorithm

MergeSort Algorithm: Overview

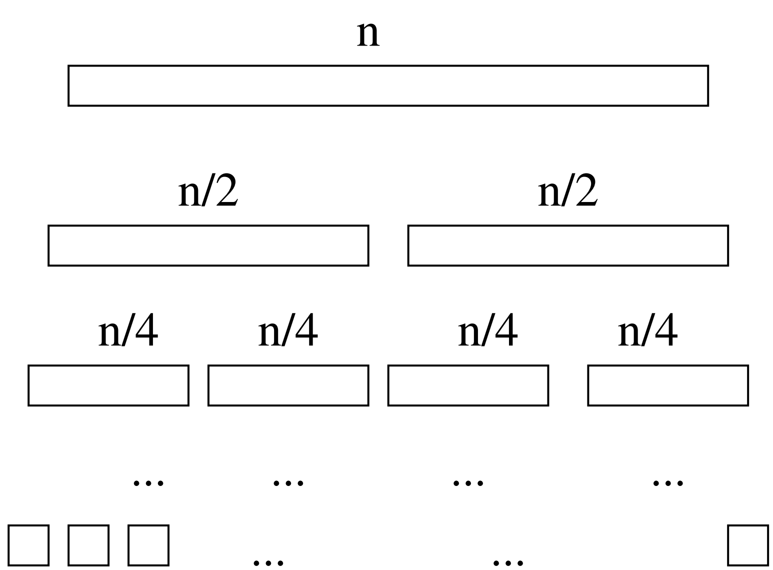
```
Algorithm MergeSort(A, lo, hi)
INPUT: A is an array of comparable elements,
  lo and hi (lo<=hi) are indices into A
OUTPUT: A[lo..hi] part of the array is sorted

1. if lo == hi then Return
2. split array into two subarrays lo-mid and (mid+1)-hi
3. MergeSort(A, lo, mid)
4. MergeSort(A, mid+1, hi)
5. Merge A[lo..mid] with A[mid+1..hi]
```

- Let us look at code

(`~prof2132/public/mergesort.c-blanks`)

Mergesort Running-Time Complexity



	Running time
level 0:	< c*n
level 1:	< 2*c*(n/2) = c*n
level 2:	< 4*c*(n/4) = c*n
...	...
level log2(n):	< n*c*1 = c*n
	Total: log2(n)*c*n

Running time complexity: $O(n \log n)$

Quicksort Algorithm

- Before we compare quicksort and mergesort, let us go through a reminder of the Quicksort algorithm

```
Algorithm Quicksort(A, lo, hi)
INPUT: A is an array of comparable elements,
  lo and hi (lo<=hi) are indices into A
OUTPUT: A[lo..hi] part of the array is sorted

if lo < hi then
  p = partition(A, lo, hi)
  quicksort(A, lo, p)
  quicksort(A, p+1, hi)
```

```
Algorithm Partition(A, lo, hi)
INPUT: A is an array of comparable elements,
  lo and hi (lo<=hi) are indices into A
OUTPUT: Index p (lo<=p<=hi) such that
  A[lo..p]<=A[p+1..hi] (each element of left sub-array
  is <= than each element of right sub-array)

pivot = A[(lo+hi)/2];   // other choices of pivot...
i = lo - 1; j = hi + 1;
while (true) do
  do i++ while A[i] < pivot
  do j-- while A[j] > pivot
  if i >= j then return j
  swap A[i] with A[j]
```

Mergesort vs. Quicksort

- Quicksort tends to be faster in practice for array sorting
- Some advantages of Mergesort:
  - Mergesort is faster for linked lists
  - Margesort can be made I/O efficient more easily for large data
  - Mergesort is easier to parallelize
  - Mergesort has better worst-case analysis ($O(n\log(n))$ vs. $O(n^2)$)

Example: Generating Permutations

- For a given positive integer n, print out all permutations of numbers $\{1, 2, \ldots, n\}$
- For example, for $n = 3$, print:

```
1 2 3
1 3 2
2 1 3
2 3 1
3 2 1
3 1 2
```

- This is a non-obvious problem and requires some thinking and algorithm design

Generating Permutations