

Merge Sort

Biostatistics 615/815

Lecture 9

Project Assignments

- Project assignments sent out by e-mail
- You have about 8 weeks to complete the project
- First step: come to office hours in the next couple of weeks and sketch out a plan for completing the project!

Scheduling ...

- We need to schedule our midterm
- Review Session
 - I propose Thursday, October 19
- Mid-term Exam
 - I propose Tuesday, October 24

Last Lecture: Quick Sort

- Choose a partitioning element ...
- Organize array such that:
 - All elements to the right are greater
 - All elements to the left are smaller
- Sort right and left sub-arrays independently

Quick Sort Summary

- Divide and Conquer Algorithm
 - Recursive calls can be “hidden”
- Optimizations
 - Choice of median
 - Threshold for brute-force methods
 - Limiting depth of recursion
- Do you think quick sort is a stable sort?

C Code: QuickSort

```
void quicksort(Item a[], int start, int stop)
{
    int i;

    if (stop <= start) return;

    i = partition(a, start, stop);
    quicksort(a, start, i - 1);
    quicksort(a, i + 1, stop);
}
```

C Code: Partitioning

```
int partition(Item a[], int start, int stop)
{
    int up = start, down = stop - 1, part = a[stop];

    if (stop <= start) return start;

    while (true)
    {
        while (isLess(a[up], part))
            up++;
        while (isLess(part, a[down]) && (up < down))
            down--;

        if (up >= down) break;
        Exchange(a[up], a[down]);
        up++; down--;
    }

    Exchange(a[up], a[stop]);
    return up;
}
```

Non-Recursive Quick Sort

```
void quicksort(Item a[], int start, int stop)
{
    int i, s = 0, stack[64];

    stack[s++] = start;
    stack[s++] = stop;

    while (s > 0)
    {
        stop = stack[--s];
        start = stack[--s];
        if (start >= stop) continue;

        i = partition(a, start, stop);
        if (i - start > stop - i)
        {
            stack[s++] = start; stack[s++] = i - 1;
            stack[s++] = i + 1; stack[s++] = stop;
        }
        else {
            stack[s++] = i + 1; stack[s++] = stop;
            stack[s++] = start; stack[s++] = i - 1;
        }
    }
}
```


Selection

- Problem of finding the k^{th} smallest value in an array
- Simple solution would involve sorting the array
 - Time proportional to $N \log N$ with Quick Sort
- Possible to improve by taking into account that only one element must fall into place
 - Time proportional to N

C Code: Selection

```
// Places kth smallest element in the kth position
// within array. Could move other elements.
void select(Item * a, int start, int stop, int k)
{
    int i;

    if (start <= stop) return;

    i = partition(a, start, stop);

    if (i > k) select(a, start, i - 1);
    if (i < k) select(a, i + 1, stop);
}
```

Merge Sort

- **Divide-And-Conquer Algorithm**
 - Divides a file in two halves
 - Merges sorted halves
- The “opposite” of quick sort
- Requires additional storage

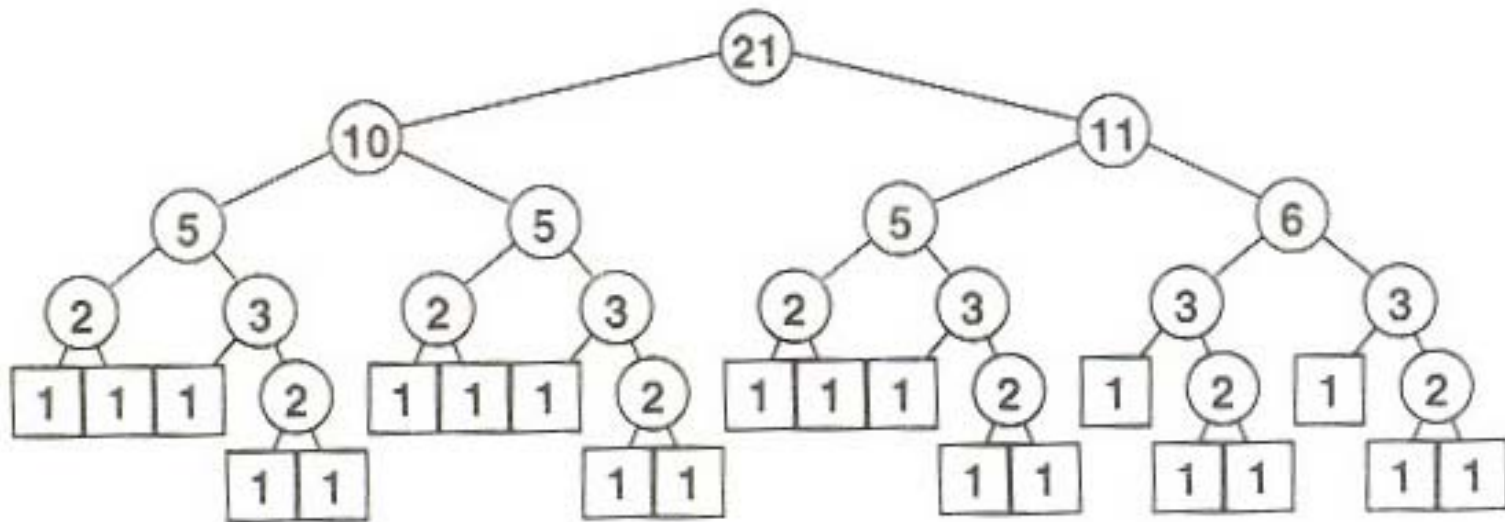
C Code: Merge Sort

```
void mergesort(Item a[], int start, int stop)
{
    int m = (start + stop)/2;

    if (stop <= start) return;

    mergesort(a, start, m);
    mergesort(a, m + 1, stop);
    merge(a, start, m, stop);
}
```

Merge Pattern N = 21



Merging Sorted Arrays

- Consider two arrays
- Assume they are both in order
- Can you think of a merging strategy?

Merging Two Sorted Arrays

```
void merge_arrays(Item merged[], Item
    a[], int N, Item b[], int M)
    {
        int i = 0, j = 0, k;

        for (k = 0; k < M + N; k++)
            {
                if (i == N)
                    { merged[k] = b[j++];
                      continue; }
            }
```

“In-Place” Merge

- For sorting, we would like to:
 - Starting with sorted halves
 - `a[start ... m]`
 - `a[m + 1 ... end]`
 - Generate a sorted stretch
 - `a[start ... end]`
- We would like an in-place merge, but...
 - A true “in-place” merge is quite complicated

Abstract In-Place Merge

- For caller, performs like in-place merge
- Creates copies two sub-arrays
- Replaces contents with merged results

C Code: Abstract In-place Merge (First Attempt)

```
Item extra1[MAX_N];
Item extra2[MAX_N];

void merge(Item a[], int start, int m, int stop)
{
    int i, j, k;

    for (i = start; i <= m; i++)
        extra1[i - start] = a[i];

    for (i = m + 1; i <= stop; i++)
        extra2[i - m - 1] = a[i];

    merge_arrays(a + start, extra1, m - start + 1,
                extra2, stop - m);
}
```

C Code: Abstract In-place Merge (Second Attempt)

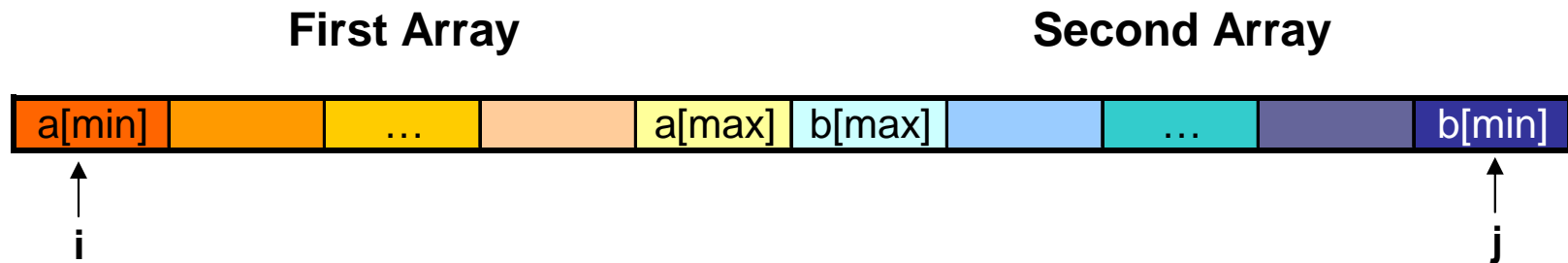
```
Item extra[MAX_N];
```

```
void merge(Item a[], int start, int m, int stop)
{
    int i, j, k;

    for (i = start; i <= stop; i++)
        extra[i] = a[i];

    for (i = k = start, j = m + 1; k <= stop; k++)
        if (j <= stop && isLess(extra[j], extra[i]) || i > m)
            a[k] = extra[j++];
        else
            a[k] = extra[i++];
}
```

Avoiding End-of-Input Check



At each point, compare elements `i` and `j`.

Then select the smallest element.

Move `i` or `j` towards the middle, as appropriate.

C Code: Abstract In-place Merge (Third Attempt!)

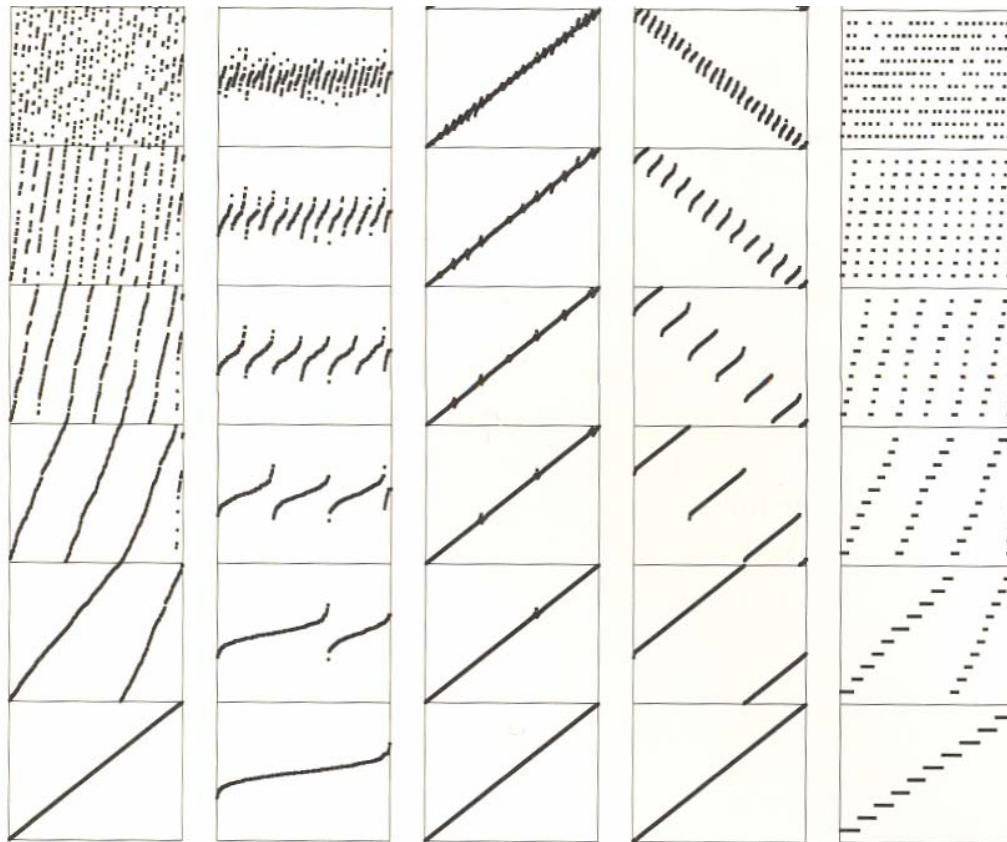
```
void merge(Item a[], int start, int m, int stop)
{
    int i, j, k;

    for (i = start; i <= m; i++)
        extra[i] = a[i];

    for (j = m + 1; j <= stop; j++)
        extra[m + 1 + stop - j] = a[j];

    for (i = k = start, j = stop; k <= stop; k++)
        if (isLess(extra[j], extra[i]))
            a[k] = extra[j--];
        else
            a[k] = extra[i++];
}
```

Merge Sort in Action



Merge Sort Notes

- Order $N \log N$
 - Number of comparisons independent of data
 - Exactly $\log N$ rounds
 - Each requires N comparisons
- Merge sort is stable
- Insertion sort for small arrays is helpful

Sedgewick's Timings (secs)

N	QuickSort	MergeSort	MergeSort*
100,000	24	53	43
200,000	52	111	92
400,000	109	237	198
800,000	241	524	426

Array of floating point numbers; * using insertion for small arrays

Non-Recursive Merge Sort

- First sort all sub-arrays of 1 element
- Perform successive merges
 - Merge results into sub-arrays of 2 elements
 - Merge results into sub-arrays of 4 elements
 - ...

Bottom-Up Merge Sort

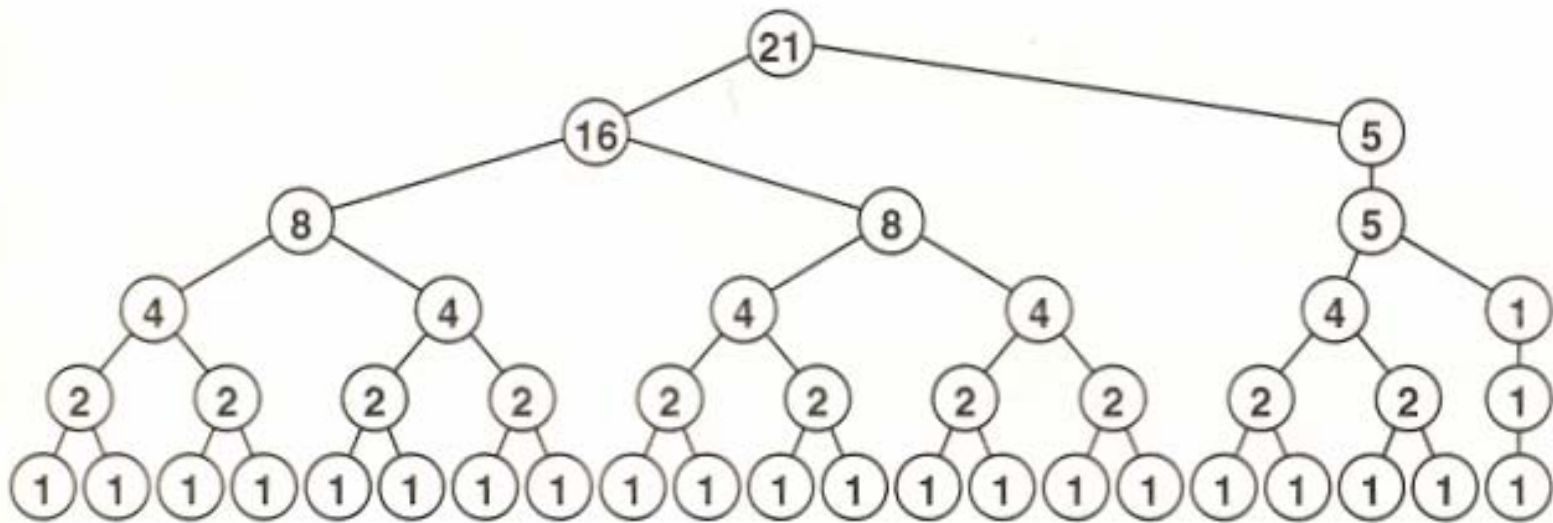
```
Item min(Item a, Item b)
  { return isLess(a,b) ? a : b; }

void mergesort(Item a[], int start, int stop)
  {
    int i, m;

    for (m = 1; m < stop - start; m += m)
      for (i = start; i < stop; i += m + m)
        {
          int from = i;
          int mid = i + m - 1;
          int to = min(i + m + m - 1, stop);

          merge(a, from, mid, to);
        }
  }
```

Merging Pattern for $N = 21$



Sedgewick's Timings (secs)

N	QuickSort	Top-Down MergeSort	Bottom-Up MergeSort
100,000	24	53	59
200,000	52	111	127
400,000	109	237	267
800,000	241	524	568

Array of floating point numbers

Automatic Memory Allocation

- Defining large static arrays is not efficient
 - Often, program will run on smaller datasets and the arrays will just waste memory
- A better way is to allocate and free memory as needed
- Create a “wrapper” function that takes care of memory allocation and freeing

Merge Sort, With Automatic Memory Allocation

```
Item * extra;

void sort(Item a[], int start, int stop)
{
    // Nothing to do with less than one element
    if (stop <= start) return;

    // Allocate the required extra storage
    extra = malloc(sizeof(Item) * (stop - start + 1));

    // Merge and sort the data
    mergesort(a, start, stop);

    // Free memory once we are done with it
    free(extra);
}
```

Today ...

- Quick Sort
- Merge Sort

- Unraveled Recursive Sorts

- Contrasting approaches to divide and conquer

Sorting Summary

- Simple $O(N^2)$ sorts for very small datasets
 - Insertion, Selection and Bubblesort
- Improved, but more complex sort
 - Shell sort
- Very efficient $N \log N$ sorts
 - Quick Sort (requires no additional storage)
 - Merge Sort (requires a bit of additional memory)

Sorting Indexes

- Generating an index is an alternative to sorting the raw data
- Allows us to keep track of many different orders
- Can be faster when items are large
- How it works:
 - Leaves the array containing the data unchanged
 - Generates an array where position i records position of the i^{th} smallest item in the original data

Example: Indexing with Insertion Sort

```
void sort(int index[], Item a[], int start, int stop)
{
    int i, j;

    for (i = start; i <= stop; i++)
        index[i] = i;

    for (i = start + 1; i <= stop; i++)
        for (j = i; j > start; j--)
            if (isLess(a[index[j]], a[index[j-1]]))
                Exchange(index[j-1], index[j])
            else
                break;
}
```

Next Lecture: An Alternative to Sorting

- We'll see how to organize data so that it can be searched ...
- And so the complexity of searching and organizing the data is less than $N \log N$
- Cost: Doing this will require additional memory

Recommended Reading

- For QuickSort
 - Sedgewick, Chapter 7
 - Hoare (1962) *Computer Journal* **5**:10-15.
- For MergeSort
 - Sedgewick, Chapter 8