CS1083 Week 9: More Recursion

David Bremner

2018-02-28

Outline

Recursive Binary Search

Parsing

Quicksort

MergeSort revisited

Recursive Binary Search

Parsing

Quicksort

MergeSort revisited

Wrapper methods

```
public int positionOf(T target) {
    if (!isSorted) {
        Collections.sort(this);
        isSorted = true;
    }
    return bSearch(target, 0, this.size()-1);
}
```

BinarySearchList2

Iteration vs. recursion

```
while (lo<=hi){
    int mid=(lo+hi)/2;
    int diff =
        get(mid)
        .compareTo(it);
    if (diff==0)
        return mid;
    if (diff<0)
        lo=mid+1;
    else
        hi=mid-1;
}
return -1;
```

```
private int bSearch(T it,int lo,
                    int hi){
    if (lo>hi) return -1;
    int mid = (lo+hi)/2;
    int diff =
      get(mid).compareTo(it);
    if (diff==0)
        return mid;
    if (diff<0)
        return bSearch(it,
                       mid+1,hi);
    else
        return bSearch(it,
                       lo,mid-1);
}
```

Recursive Binary Search

- Add tracing

```
private int bSearch(T it,int left,int right){
    if (left>right)
        return -1;
    int mid = (left+right)/2;
    int diff =
      get(mid).compareTo(it);
    if (diff==0)
        return mid;
    if (diff<0)
        return bSearch(it,mid+1,right);
    else
        return bSearch(it,left,mid-1);
}
```

Recursive Binary Search

Parsing

Quicksort

MergeSort revisited

Grammars

- Formal languages (like Java!) are often described by grammars.
- Grammars are naturally recursive.
- Here is the grammar for a simple arithmetic expression:

```
expression -> ( expression )
expression -> expression op expression
expression -> number
op -> + | - | * | /
```

Easy direction: generate some expressions

```
expression -> expression op expression
           -> expression +  expression
           -> 100+32
```

Digression: StreamTokenizer

```
StreamTokenizer st=new StreamTokenizer(new FileReader("table
int type=st.nextToken();
while (type!= StreamTokenizer.TT_EOF) {
    switch(type){
    case StreamTokenizer.TT_WORD:
        System.out.println("word: "+st.sval);
        break;
    case StreamTokenizer.TT_NUMBER:
        System.out.println("number: "+st.nval);
        break;
    }
    type=st.nextToken();
}
```

StreamTokenizerExample

Try to match the three possibilities in order

```
expression -> ( expression )
expression -> expression op expression
expression -> number
```

StringParser2

```
private double eval(int level) {
    double left;
    int token=tokens.nextToken();
    if (token=='('){
        left=eval(level+1);
        if (tokens.nextToken()!=')') die("expected ')'");
    } else if (token==StreamTokenizer.TT_NUMBER)
        left=tokens.nval;
    else
        die("Syntax error");
}
```

Try each operator

```
    double right=0.0;
    int op=tokens.nextToken();
    if (op=='+' || op=='-' || op=='*' || op=='/'){
      // rule 2
      right=eval(level);
      return arith(level,op,left,right);
    } else {
      // oops, we read too far.
      // Turns out to be rule 3
      tokens.pushBack();
      return left;
    }

}
```

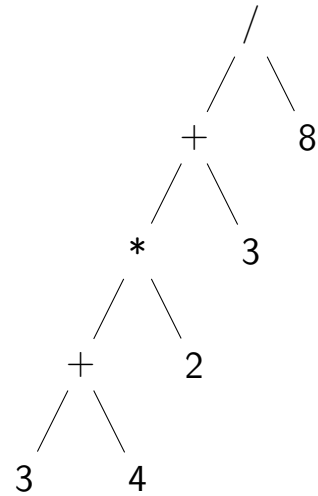
Example

```
StringParser2 Parser=
      new StringParser2(
        "(((3+4)*2)+3)/8)");

double d=Parser.eval(0);
```

```
enter eval
  enter eval
    enter eval
      enter eval
        enter eval
          enter eval
          return 4.0
        return 3.0+4.0=7.0
        enter eval
        return 2.0
      return 7.0*2.0=14.0
      enter eval
      return 3.0
```

Example

```
StringParser2 Parser=
      new StringParser2(
        "(((3+4)*2)+3)/8)");

double d=Parser.eval(0);
```

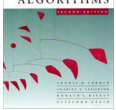


Recursive Binary Search

Parsing

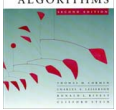
Quicksort

MergeSort revisited



Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).



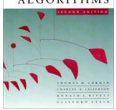
Divide and conquer

Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

Key: *Linear-time partitioning subroutine.*

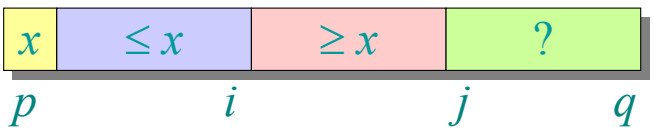


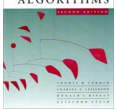
Partitioning subroutine

```
PARTITION(A, p, q)   ▷ A[p . . q]
    x ← A[p]         ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
                then  i ← i + 1
                      exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

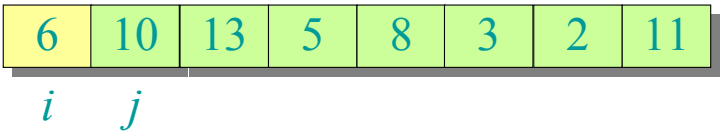
Running time $= O(n)$ for n elements.

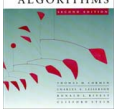
Invariant:



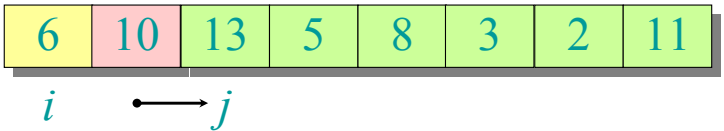


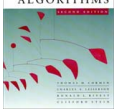
Example of partitioning



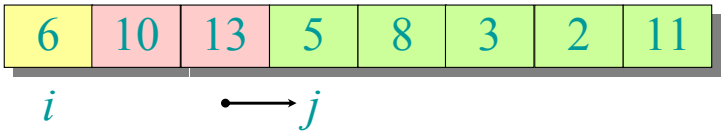


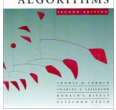
Example of partitioning



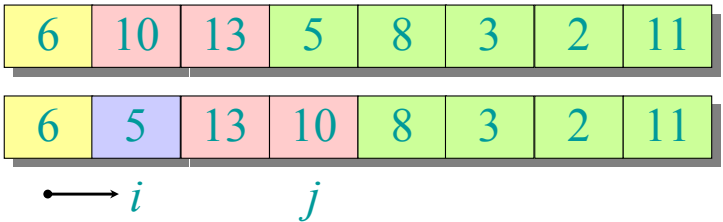


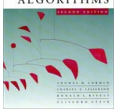
Example of partitioning



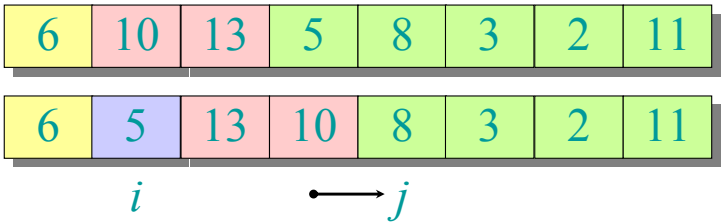


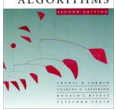
Example of partitioning



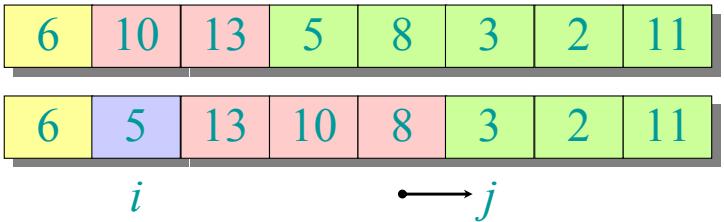


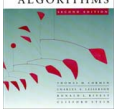
Example of partitioning



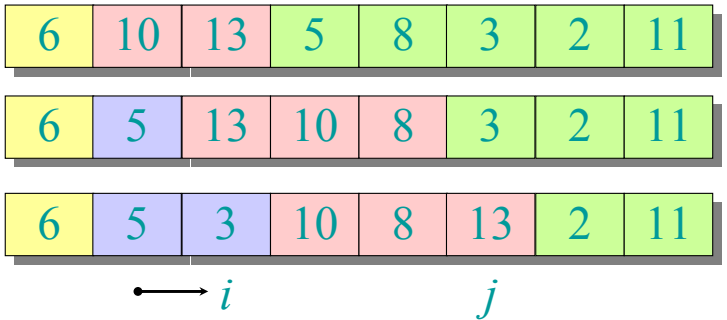


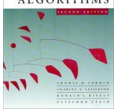
Example of partitioning



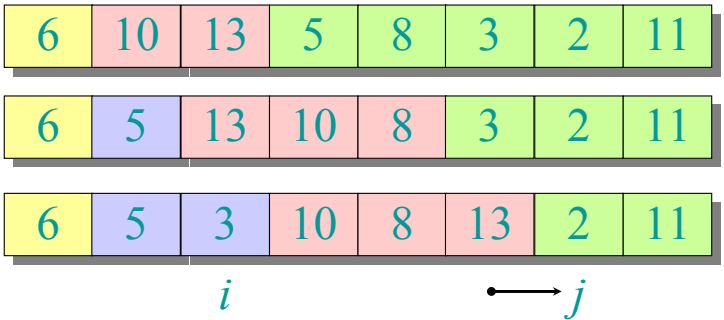


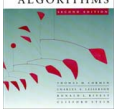
Example of partitioning



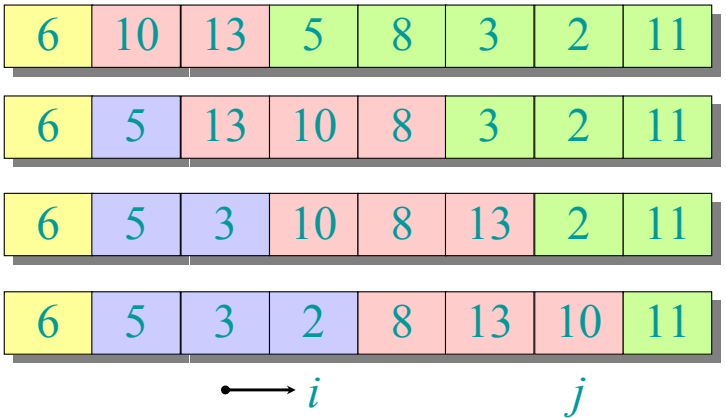


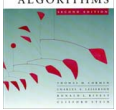
Example of partitioning



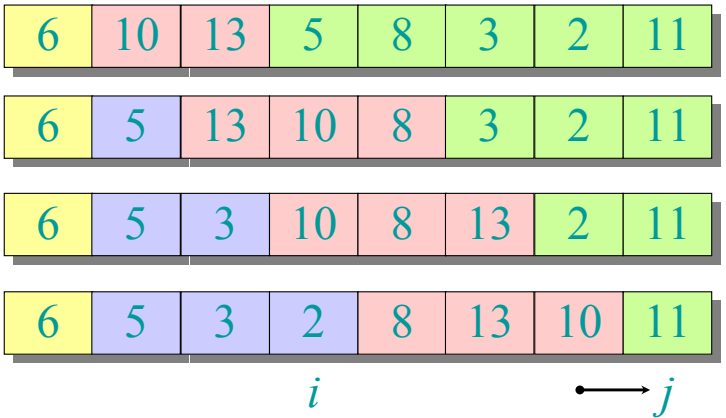


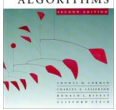
Example of partitioning



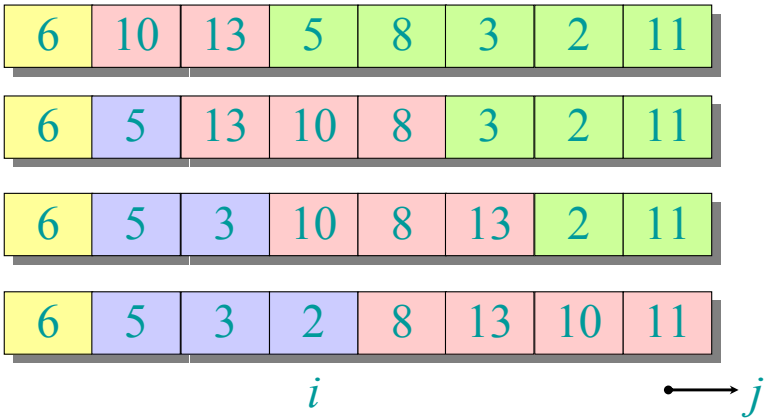


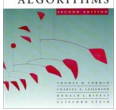
Example of partitioning





Example of partitioning

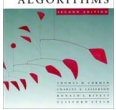




Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
2	5	3	6	8	13	10	11

i



Pseudocode for quicksort

```
QUICKSORT(A, p, r)
    if p < r
        then q ← PARTITION(A, p, r)
             QUICKSORT(A, p, q–1)
             QUICKSORT(A, q+1, r)
```

Initial call: QUICKSORT$(A, 1, n)$

Quicksort in Java

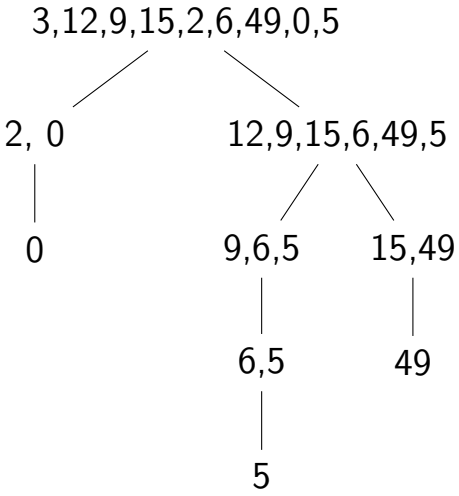
```
public static void quickSort(int[] a, int from, int to){

    if (from >= to) return;

    int mid=partition(a,from,to);

    quickSort(a, from, mid-1);
    quickSort(a, mid+1, to);
}
```

QuickSort

Partition in Java

```
private static int partition(int[] a,int from,int to){
    int pivot = a[from];
    int i = from;
    for (int j=from+1; j<= to; j++)
        if(a[j] <= pivot ) {
            i++;
            int temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
    a[from]=a[i];
    a[i]=pivot;
    return i;
}
```

Quicksort example



- Here we choose pivot from first element; how could we do better?

Recursive Binary Search

Parsing

Quicksort

MergeSort revisited

Replacing Bottom levels of MergeSort

MyMergeSort2

```
public static void sort(int[] a){
    if (a.length <= 1) return;
    if (a.length <= 26){
        IntSelectionSort.sort(a);
    } else {









    }
}
```

Replacing Bottom levels of MergeSort

MyMergeSort2

```
    public static void sort(int[] a){
        if (a.length <= 1) return;
        if (a.length <= 26){
            IntSelectionSort.sort(a);
        } else {
            int mid=a.length/2;
            int[] left=Arrays.copyOfRange(a,0,mid);
            int[] right=Arrays.copyOfRange(a,mid,
                                           a.length);



        }
    }
```

Replacing Bottom levels of MergeSort

MyMergeSort2

```
public static void sort(int[] a){
    if (a.length <= 1) return;
    if (a.length <= 26){
        IntSelectionSort.sort(a);
    } else {
        int mid=a.length/2;
        int[] left=Arrays.copyOfRange(a,0,mid);
        int[] right=Arrays.copyOfRange(a,mid,
                                       a.length);
        sort(left); sort(right);
        merge(left,right,a);
    }
}
```

Comparing Sort algorithms

```
for (int len=increment; len <= steps*increment; len+=increme
    int[] a = ArrayUtil.randomIntArray(len, 1000000);
    switch(sort){
    case 0:
        clock.start();
        IntSelectionSort.sort(a);
        clock.stop();
        break;
    case 1:
        clock.start();
        MyMergeSort.sort(a);
        clock.stop();
        break;
    case 2:
        clock.start();
        MergeSort.sort(a);
```

SortTest

A clock utility class

StopWatch

```
public void start()
{
    if (isRunning) return;
    isRunning = true;
    startTime = System.nanoTime();
}

public void stop()
{
    if (!isRunning) return;
    isRunning = false;
    long endTime = System.nanoTime();
    elapsedTime = elapsedTime + endTime - startTime;
}
```