CS1083 Week 8: Recursion

David Bremner

2018-02-21

Outline

Recursion

Mazes

Towers of Hanoi

Recursion with objects

Merge Sort

Recursion

Bottom Up To compute the next element of a sequence, apply some operation to the previous elements.

Top Down To solve a problem for size n, use solutions for same problem of size $k < n$.

Recursion

Bottom Up To compute the next element of a sequence, apply some operation to the previous elements.

Top Down To solve a problem for size n, use solutions for same problem of size $k < n$.

Factorial

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

recursive definition

$$n! = n \times (n-1)! \tag{1}$$

- Try to compute 5! using (1)
- What are we missing?

Factorial

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

recursive definition

$$n! = n \times (n-1)! \tag{1}$$

- Try to compute 5! using (1)
- What are we missing?

Factorial

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

recursive definition

$$n! = n \times (n-1)! \tag{1}$$

- Try to compute 5! using (1)
- What are we missing?

Factorial

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

recursive definition

$$n! = n \times (n-1)! \tag{1}$$

- Try to compute 5! using (1)
- What are we missing?

Factorial in Java


```
static int factorial(int n){
    int return_val;
    Trace.reverseIndent(n,
        "entering factorial("+n+")");









}
```

Factorial in Java


```
static int factorial(int n){
    int return_val;
    Trace.reverseIndent(n,
        "entering factorial("+n+")");
    if (n<=1)
        return_val=1;
    else
        return_val=n*factorial(n-1);



}
```

Factorial in Java


```
static int factorial(int n){
    int return_val;
    Trace.reverseIndent(n,
        "entering factorial("+n+")");
    if (n<=1)
        return_val=1;
    else
        return_val=n*factorial(n-1);
    Trace.reverseIndent(n,
        "factorial(" +n+")="+ return_val);
    return return_val;
}
```

Factorial Trace

```
 entering factorial(6)
  entering factorial( 5 )
    entering factorial(4)
      entering factorial(3)
        entering factorial(2)
          entering factorial(1)
          factorial(1)=1
        factorial(2)=2
      factorial(3)=6
    factorial(4)=
  factorial(5)=120
factorial(6)=720
```

Factorial Trace

```
 entering factorial(6)
  entering factorial( 5 )
    entering factorial(4)
      entering factorial(3)
        entering factorial(2)
          entering factorial(1)
          factorial(1)=1
        factorial(2)=2
      factorial(3)=6
    factorial(4)= 24
  factorial(5)=120
factorial(6)=720
```

Fibonacci Function

$$F_1 = 1$$
$$F_2 = 1$$
$$F_i = F_{i-1} + F_{i-2}$$

- Compute F_6.

Fibonacci in Java

FibTrace

```
public static int fib(int n, int level){
    Trace.indent(level,"Entering fib("+ n+")");









    return f;
}
```

Fibonacci in Java

FibTrace

```
public static int fib(int n, int level){
    Trace.indent(level,"Entering fib("+ n+")");
    int f;
    if (n <= 2)
        f = 1;
    else
        f=fib(n-1,level+1)+fib(n-2,level+1);

    return f;
}
```

Fibonacci in Java

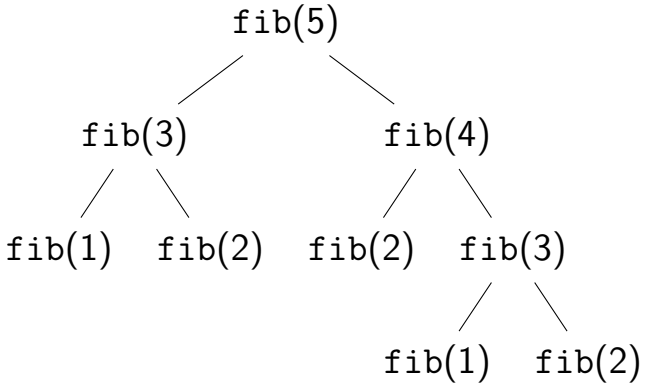
FibTrace

```
public static int fib(int n, int level){
    Trace.indent(level,"Entering fib("+ n+")");
    int f;
    if (n <= 2)
        f = 1;
    else
        f=fib(n-1,level+1)+fib(n-2,level+1);
    Trace.indent(level,"fib("+n+")= "+f);
    return f;
}
```

Fibonacci Trace

```
Entering fib(4)
  Entering fib(3)
    Entering fib(2)
    fib(2)= 1
    Entering fib(1)
    fib(1)= 1
  fib(3)= 2
  Entering fib(2)
  fib(2)= 1
fib(4)= 3
```

Fibonacci calls



Fibonacci discussion

- Compare figure to our hand computation.
- Which is more efficient?
- The efficiency of recursion depends on
  - How many subproblems
  - How big each subproblem is.

Converting strings to integers

$$\text{num}(s) = \begin{cases} s & |s| == 1 \\ 10 \times \text{num}(\text{start}(s)) + \text{last}(s) & \text{otherwise} \end{cases}$$

where

$$\text{last}(abc) = c$$
$$\text{start}(abc) = ab$$

- Try num("103")

in Java

EvalNumber2

```
private static long evalNumber(String digits){
  int  len=digits.length() ;
  if (len==1)
                          ;
  return 10*evalNumber(                    )+
         evalDigit(digits.substring(len-1));

}
```

- What is the advantage of peeling of characters on the right?
- How to implement `evalDigit`?

in Java

EvalNumber2

```
private static long evalNumber(String digits){
  int  len=digits.length() ;
  if (len==1)
     return evalDigit(digits) ;
  return 10*evalNumber(                    )+
         evalDigit(digits.substring(len-1));

}
```

- What is the advantage of peeling of characters on the right?
- How to implement `evalDigit`?

in Java

EvalNumber2

```
private static long evalNumber(String digits){
  int  len=digits.length() ;
  if (len==1)
     return evalDigit(digits) ;
  return 10*evalNumber( digits.substring(0,len-1) )+
         evalDigit(digits.substring(len-1));

}
```

- What is the advantage of peeling of characters on the right?
- How to implement `evalDigit`?

in Java

EvalNumber2

```
private static long evalNumber(String digits){
  int  len=digits.length() ;
  if (len==1)
     return evalDigit(digits) ;
  return 10*evalNumber( digits.substring(0,len-1) )+
         evalDigit(digits.substring(len-1));

}
```

- What is the advantage of peeling of characters on the right?
- How to implement `evalDigit`?

in Java

EvalNumber2

```
private static long evalNumber(String digits){
  int  len=digits.length() ;
  if (len==1)
     return evalDigit(digits) ;
  return 10*evalNumber( digits.substring(0,len-1) )+
         evalDigit(digits.substring(len-1));

}
```

- What is the advantage of peeling of characters on the right?
- How to implement `evalDigit`?

Reversing Strings

$$\text{reverse}(s) = \begin{cases} s & |s| == 1 \\ \text{reverse}(\text{end}(s)) + \text{first}(s) & \text{otherwise} \end{cases}$$

where

$$\text{first}(abc) = a$$
$$\text{end}(abc) = bc$$

- Try reverse(cat)

Reversing in java

RecurseReverse

```
public static String reverse(String s){
    String rval;
    Trace.reverseIndent(s.length(),
        "entering reverse("+s+")");




    Trace.reverseIndent(s.length(),
        "reverse("+s+")="+rval);
    return rval;
}
```

Reversing in java

RecurseReverse

```
public static String reverse(String s){
    String rval;
    Trace.reverseIndent(s.length(),
         "entering reverse("+s+")");
    if (s.length()<=1)
        rval=s;
    else
        rval=reverse(s.substring(1))+s.charAt(0);
    Trace.reverseIndent(s.length(),
         "reverse("+s+")="+rval);
    return rval;
}
```

Reversing Trace

```
entering  reverse(scoobydoo)
  entering reverse(       )
    entering reverse(oobydoo)
                ⋮
               entering reverse(oo)
                 entering reverse(o)
                 reverse(o)=o
               reverse(oo)=oo
             reverse(doo)=ood
           reverse(    )=
         reverse(bydoo)=oodyb
         ⋮
reverse(scoobydoo)=oodyboocs
```

Reversing Trace

```
entering  reverse(scoobydoo)
  entering reverse( coobydoo )
    entering reverse(oobydoo)
                ⋮
               entering reverse(oo)
                 entering reverse(o)
                 reverse(o)=o
               reverse(oo)=oo
             reverse(doo)=ood
           reverse(     )=
         reverse(bydoo)=oodyb
         ⋮
reverse(scoobydoo)=oodyboocs
```

Reversing Trace

```
entering  reverse(scoobydoo)
  entering reverse( coobydoo )
    entering reverse(oobydoo)
                ⋮
               entering reverse(oo)
                 entering reverse(o)
                 reverse(o)=o
               reverse(oo)=oo
             reverse(doo)=ood
           reverse( ydoo )=
         reverse(bydoo)=oodyb
         ⋮
reverse(scoobydoo)=oodyboocs
```

Reversing Trace

```
entering  reverse(scoobydoo)
  entering reverse( coobydoo )
    entering reverse(oobydoo)
                ⋮
               entering reverse(oo)
                 entering reverse(o)
                 reverse(o)=o
               reverse(oo)=oo
             reverse(doo)=ood
           reverse( ydoo )= oody
         reverse(bydoo)=oodyb
         ⋮
reverse(scoobydoo)=oodyboocs
```

Recursion

Mazes

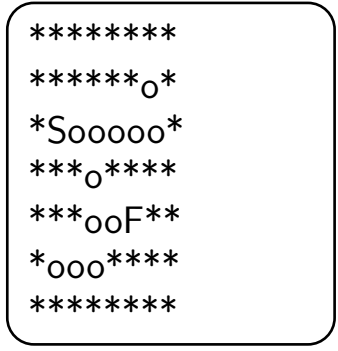
Towers of Hanoi

Recursion with objects

Merge Sort

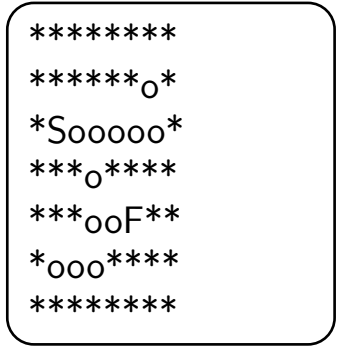
Searching in a maze

- Maze stored in 2D array
- Cells are marked wall, empty, explored, path, start, finish.
- Want to go from start only on *path* squares
- Assume maze is bounded by walls.



Searching in a maze

- Maze stored in 2D array
- Cells are marked wall, empty, explored, path, start, finish.
- Want to go from start only on *path* squares
- Assume maze is bounded by walls.



Searching in a maze

- Maze stored in 2D array
- Cells are marked wall, empty, explored, path, start, finish.
- Want to go from start only on *path* squares
- Assume maze is bounded by walls.

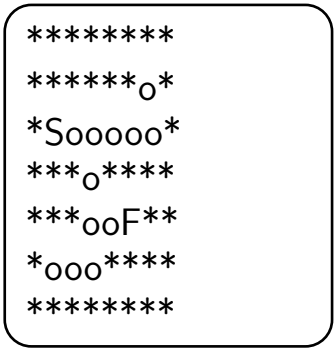
```
********
******o*
*Sooooo*
***o****
***ooF**
*ooo****
********
```

Searching in a maze

- Maze stored in 2D array
- Cells are marked wall, empty, explored, path, start, finish.
- Want to go from start only on *path* squares
- Assume maze is bounded by walls.

```
********
******o*
*Sooooo*
***o****
***ooF**
*ooo****
********
```

Searching recursively



Recursive Idea:

There is a path from the current cell to *Finish* if there is a path from one of the neighbours.

Gotcha:

How to keep from going in circles?

Searching recursively

```
********
******o*
*Sooooo*
***o****
***ooF**
*ooo****
********
```

Recursive Idea:

There is a path from the current cell to *Finish* if there is a path from one of the neighbours.

Gotcha:

How to keep from going in circles?

Searching in a maze

Maze

```
public boolean path(){
    boolean res = path(startr, startc);
    map[startr][startc] = Cell.START;
    return res;
}
```

Searching in a maze

```
private boolean path(int r, int c){
    if(map[r][c] == Cell.FINISH)
        return true;
```

Searching in a maze

```
private boolean path(int r, int c){
    if(map[r][c] == Cell.FINISH)
        return true;
    if((map[r][c] == Cell.EXPLORED) ||
       (map[r][c] == Cell.WALL))
        return false;
```

Searching in a maze

```
private boolean path(int r, int c){
    if(map[r][c] == Cell.FINISH)
        return true;
    if((map[r][c] == Cell.EXPLORED) ||
       (map[r][c] == Cell.WALL))
        return false;
    map[r][c] = Cell.EXPLORED;
```

Searching in a maze

```
    int [][] dirs = {{-1,0}, {1,0},
                     {0,-1}, {0,1}};
    for (int d = 0; d<4; d++) {
        int i = r+dirs[d][0];
        int j = c+dirs[d][1];
        if(path(i,j)){
            map[r][c] = Cell.FOUND;
            return true;
        }
    }
    return false;
}
```

Recursion

Mazes

Towers of Hanoi

Recursion with objects

Merge Sort

Towers of Hanoi

- Three pegs
- all disks on first peg.
- want to move to third peg.
- smaller on larger.

Problem decomposition

Notice if we can get the top $k-1$ disks to peg 2, we can then move the last disk.

We are then left with problem of moving $k-1$ disks from peg 2 to peg 3.

Towers of Hanoi

- Three pegs
- all disks on first peg.
- want to move to third peg.
- smaller on larger.

Problem decomposition

Towers of Hanoi

- Three pegs
- all disks on first peg.
- want to move to third peg.
- smaller on larger.

Problem decomposition

- Notice if we can get the top $k-1$ disks to peg 2, we can then move the last disk.
- We are then left with problem of moving $k-1$ disks from peg 2 to peg 3.

Towers of Hanoi

- Three pegs
- all disks on first peg.
- want to move to third peg.
- smaller on larger.

Problem decomposition

- Notice if we can get the top $k-1$ disks to peg 2, we can then move the last disk.
- We are then left with problem of moving $k-1$ disks from peg 2 to peg 3.

Towers of Hanoi

- Three pegs
- all disks on first peg.
- want to move to third peg.
- smaller on larger.

Problem decomposition

- Notice if we can get the top $k-1$ disks to peg 2, we can then move the last disk.
- We are then left with problem of moving $k-1$ disks from peg 2 to peg 3.

Towers of Hanoi

- Three pegs
- all disks on first peg.
- want to move to third peg.
- smaller on larger.

Problem decomposition

- Notice if we can get the top $k-1$ disks to peg 2, we can then move the last disk.
- We are then left with problem of moving $k-1$ disks from peg 2 to peg 3.

Towers of Hanoi

- Three pegs
- all disks on first peg.
- want to move to third peg.
- smaller on larger.

Problem decomposition

- Notice if we can get the top $k-1$ disks to peg 2, we can then move the last disk.
- We are then left with problem of moving $k-1$ disks from peg 2 to peg 3.

move k disks from peg a to peg b using peg c

preconditions

- peg a has at least k disks.
- all disks on pegs b and c are bigger than those on peg a.


```
public static void solve(int disks, int from,
                         int to){
```

move k disks from peg a to peg b using peg c

preconditions

- peg a has at least k disks.
- all disks on pegs b and c are bigger than those on peg a.


```
public static void solve(int disks, int from,
                         int to){
```

move k disks from peg a to peg b using peg c

preconditions

- peg a has at least k disks.
- all disks on pegs b and c are bigger than those on peg a.

Hanoi

```
public static void solve(int disks, int from,
                         int to){
    Trace.reverseIndent(disks, "solve(disks="+
        disks + ",from=" +from+ ",to=" +to+")");
```

move k disks from peg a to peg b using peg c


```
public static void solve(int disks, int from,
                         int to){
    Trace.reverseIndent(disks, "solve(disks="+
        disks + ",from=" +from+ ",to=" +to+")");
    if (disks==1){
        printmove(1,from,to);
    } else {
        int using=6-from-to;




    }
```

move k disks from peg a to peg b using peg c


```
public static void solve(int disks, int from,
                         int to){
    Trace.reverseIndent(disks, "solve(disks="+
        disks + ",from=" +from+ ",to=" +to+")");
    if (disks==1){
        printmove(1,from,to);
    } else {
        int using=6-from-to;
        solve(disks-1,from,using);
        printmove(disks,from,to);
        solve(disks-1,using,to);
    }
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
    solve(disks=2, from=1, to=3)




    *Move disk from 1 to 2
    solve(disks=2, from=3, to=2)
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
    solve(disks=2, from=1, to=3)


      *Move disk from 1 to 3


    *Move disk from 1 to 2
    solve(disks=2, from=3, to=2)
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
    solve(disks=2, from=1, to=3)
      solve(disks=1, from=1, to=2)
        *Move disk from 1 to 2
      *Move disk from 1 to 3


    *Move disk from 1 to 2
    solve(disks=2, from=3, to=2)
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
    solve(disks=2, from=1, to=3)
      solve(disks=1, from=1, to=2)
        *Move disk from 1 to 2
      *Move disk from 1 to 3
      solve(disks=1, from=2, to=3)
        *Move disk from 2 to 3
    *Move disk from 1 to 2
    solve(disks=2, from=3, to=2)
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
    solve(disks=2, from=1, to=3)
      solve(disks=1, from=1, to=2)
        *Move disk from 1 to 2
      *Move disk from 1 to 3
      solve(disks=1, from=2, to=3)
        *Move disk from 2 to 3
    *Move disk from 1 to 2
    solve(disks=2, from=3, to=2)
      solve(disks=1, from=3, to=1)
        *Move disk from 3 to 1
      *Move disk from 3 to 2
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
    solve(disks=2, from=1, to=3)
      solve(disks=1, from=1, to=2)
        *Move disk from 1 to 2
      *Move disk from 1 to 3
      solve(disks=1, from=2, to=3)
        *Move disk from 2 to 3
    *Move disk from 1 to 2
    solve(disks=2, from=3, to=2)
      solve(disks=1, from=3, to=1)
        *Move disk from 3 to 1
      *Move disk from 3 to 2
```

4 disk solution

```
solve(disks=4, from=1, to=3)
  solve(disks=3, from=1, to=2)
  *Move disk from 1 to 3
  solve(disks=3, from=2, to=3)
    solve(disks=2, from=2, to=1)
      solve(disks=1, from=2, to=3)
        *Move disk from 2 to 3
      *Move disk from 2 to 1
      solve(disks=1, from=3, to=1)
        *Move disk from 3 to 1
    *Move disk from 2 to 3
    solve(disks=2, from=1, to=3)
      solve(disks=1, from=1, to=2)
```

Recursion

Mazes

Towers of Hanoi

Recursion with objects

Merge Sort

Recursion with objects

ObjFact

```
public ObjFact(int n){
    this.n=n;
    child = null;
}
```

Recursion with objects

ObjFact

```
public ObjFact(int n){
    this.n=n;
    child = null;
}
public int fact(){









}
```

Recursion with objects

ObjFact

```
public ObjFact(int n){
    this.n=n;
    child = null;
}
public int fact(){
    if (n<=1) {

    } else {



    }
}
```

Recursion with objects

ObjFact

```
public ObjFact(int n){
    this.n=n;
    child = null;
}
public int fact(){
    if (n<=1) {
        return 1;
    } else {



    }
}
```

Recursion with objects

ObjFact

```
public ObjFact(int n){
    this.n=n;
    child = null;
}
public int fact(){
    if (n<=1) {
        return 1;
    } else {
        child=new ObjFact(n-1);

    }
}
```

Recursion with objects

ObjFact

```
public ObjFact(int n){
    this.n=n;
    child = null;
}
public int fact(){
    if (n<=1) {
        return 1;
    } else {
        child=new ObjFact(n-1);
        return n* child.fact();
    }
}
```

Merge Sort

Sort array A of n elements as follows:

1. If A has 2 or more elements, split A into
   - A_1 - containing $\left\lceil \frac{n}{2} \right\rceil$ elements
   - A_2 - containing $\left\lfloor \frac{n}{2} \right\rfloor$ elements
2. Recursively sort A_1 and A_2
3. Merge A_1 & A_2 back into A

(14	3	12	17	6	2	17	8)	split
(14	3	12	17)	(6	2	17	8)	split
(14	3)	(12	17)	(6	2)	(17	8)	split
(14)	(3)	(12)	(17)	(6)	(2)	(17)	(8)	solve!
(3	14)	(	17)	(2	6)	(8	17)	merge
(3	12	14	)	(2	6	8	)	merge
(2	3	6	8	12	14	17	17)	done!

(14	3	12	17	6	2	17	8)	split
(14	3	12	17)	(6	2	17	8)	split
(14	3)	(12	17)	(6	2)	(17	8)	split
(14)	(3)	(12)	(17)	(6)	(2)	(17)	(8)	solve!
(3	14)	(12	17)	(2	6)	(8	17)	merge
(3	12	14	)	(2	6	8	)	merge
(2	3	6	8	12	14	17	17)	done!

(14	3	12	17	6	2	17	8)	split
(14	3	12	17)	(6	2	17	8)	split
(14	3)	(12	17)	(6	2)	(17	8)	split
(14)	(3)	(12)	(17)	(6)	(2)	(17)	(8)	solve!
(3	14)	(12	17)	(2	6)	(8	17)	merge
(3	12	14	17)	(2	6	8	)	merge
(2	3	6	8	12	14	17	17)	done!

(14	3	12	17	6	2	17	8)	split
(14	3	12	17)	(6	2	17	8)	split
(14	3)	(12	17)	(6	2)	(17	8)	split
(14)	(3)	(12)	(17)	(6)	(2)	(17)	(8)	solve!
(3	14)	(12	17)	(2	6)	(8	17)	merge
(3	12	14	17)	(2	6	8	17)	merge
(2	3	6	8	12	14	17	17)	done!

Merge Sort with Copying

MyMergeSort

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




}
```

Merge Sort with Copying

MyMergeSort

```
public static void sort(int[] a){
    if (a.length <= 1)
        return;
    int mid=a.length/2;









}
```

Merge Sort with Copying

MyMergeSort

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




}
```

Merge Sort with Copying

MyMergeSort

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

Merge Sort with Copying

```
public static void
    merge(int[] a, int[] b, int[] out){
    int aIndex=0, bIndex=0;
```

Merge Sort with Copying

```
public static void
    merge(int[] a, int[] b, int[] out){
    int aIndex=0, bIndex=0;
    for (int i=0; i<out.length; i++){




    }
```

Merge Sort with Copying

```
public static void
    merge(int[] a, int[] b, int[] out){
    int aIndex=0, bIndex=0;
    for (int i=0; i<out.length; i++){




    }
```

Merge Sort with Copying

```
public static void
    merge(int[] a, int[] b, int[] out){
    int aIndex=0, bIndex=0;
    for (int i=0; i<out.length; i++){
        if(aIndex >= a.length)
            out[i]=b[bIndex++];
```

```
    }
```

Merge Sort with Copying

```
public static void
    merge(int[] a, int[] b, int[] out){
    int aIndex=0, bIndex=0;
    for (int i=0; i<out.length; i++){
        if(aIndex >= a.length)
            out[i]=b[bIndex++];
        else if (bIndex >= b.length)
            out[i]=a[aIndex++];
```

```
    }
```

Merge Sort with Copying

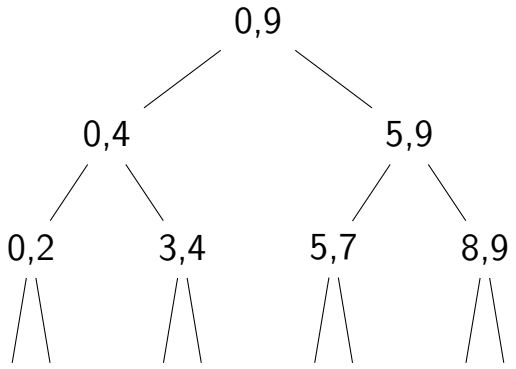
```
public static void
    merge(int[] a, int[] b, int[] out){
    int aIndex=0, bIndex=0;
    for (int i=0; i<out.length; i++){
        if(aIndex >= a.length)
            out[i]=b[bIndex++];
        else if (bIndex >= b.length)
            out[i]=a[aIndex++];
        else if (a[aIndex] <= b[bIndex])
            out[i]=a[aIndex++];


    }
```

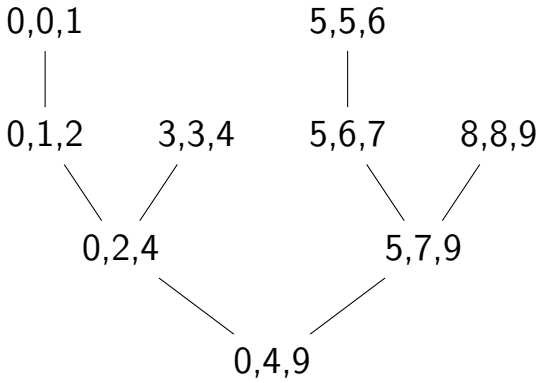
Merge Sort with Copying

```
public static void
    merge(int[] a, int[] b, int[] out){
    int aIndex=0, bIndex=0;
    for (int i=0; i<out.length; i++){
        if(aIndex >= a.length)
            out[i]=b[bIndex++];
        else if (bIndex >= b.length)
            out[i]=a[aIndex++];
        else if (a[aIndex] <= b[bIndex])
            out[i]=a[aIndex++];
        else
            out[i]=b[bIndex++];
    }
```

Splitting via indices (low,high) for n=10



Merge calls, by index



(Mostly) in place merge sort

MergeSort

```
public static void mergeSort(int[] a, int from,
                             int to){
    if (from == to) return;
    int mid = (from + to) / 2;




}
```

(Mostly) in place merge sort

MergeSort

```
public static void mergeSort(int[] a, int from,
                             int to){
    if (from == to) return;
    int mid = (from + to) / 2;

    mergeSort(a, from, mid);
    mergeSort(a, mid + 1, to);


}
```

(Mostly) in place merge sort

MergeSort

```
public static void mergeSort(int[] a, int from,
                             int to){
    if (from == to) return;
    int mid = (from + to) / 2;

    mergeSort(a, from, mid);
    mergeSort(a, mid + 1, to);

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

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];









    }

}
```

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];
    int aIndex=from;          int bIndex=mid+1;









    }

}
```

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];
    int aIndex=from;          int bIndex=mid+1;
    for (int i=0; i<dest.length; i++){









    }


}
```

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];
    int aIndex=from;        int bIndex=mid+1;
    for (int i=0; i<dest.length; i++){
        if(aIndex > mid) dest[i]=a[bIndex++];









    }


}
```

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];
    int aIndex=from;        int bIndex=mid+1;
    for (int i=0; i<dest.length; i++){
        if(aIndex > mid) dest[i]=a[bIndex++];
        else if (bIndex > to) dest[i]=a[aIndex++];




    }


}
```

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];
    int aIndex=from;        int bIndex=mid+1;
    for (int i=0; i<dest.length; i++){
        if(aIndex > mid) dest[i]=a[bIndex++];
        else if (bIndex > to) dest[i]=a[aIndex++];
        else if (a[aIndex] <= a[bIndex])
            dest[i]=a[aIndex++];

    }

}
```

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];
    int aIndex=from;        int bIndex=mid+1;
    for (int i=0; i<dest.length; i++){
        if(aIndex > mid) dest[i]=a[bIndex++];
        else if (bIndex > to) dest[i]=a[aIndex++];
        else if (a[aIndex] <= a[bIndex])
            dest[i]=a[aIndex++];
        else dest[i]=a[bIndex++];
    }

}
```

(Mostly) in place merge sort

```
public static void merge(int[] a,int from,int mid,int to){
    int[] dest = new int[to-from+1];
    int aIndex=from;        int bIndex=mid+1;
    for (int i=0; i<dest.length; i++){
        if(aIndex > mid) dest[i]=a[bIndex++];
        else if (bIndex > to) dest[i]=a[aIndex++];
        else if (a[aIndex] <= a[bIndex])
            dest[i]=a[aIndex++];
        else dest[i]=a[bIndex++];
    }
    System.arraycopy(dest,0,a,0,dest.length);
}
```

Analysis of merge sort

- How many elements in each layer of the recursion tree
- How many layers in the tree?
- Let n_i be $\texttt{high} - \texttt{low}$ after i iterations.
- How many times can we divide n by 2 before we get 1?

$$
\begin{aligned}
n_0 &= n \\
n_i &< \frac{n_{i-1}}{2} \qquad \text{(why?)} \\
&< \frac{n_{i-2}}{4} \\
&\vdots \\
&< \frac{n}{2^i}
\end{aligned}
$$