12 A: Algorithm Design Techniques II

CS1102S: Data Structures and Algorithms

Martin Henz

April 7, 2010

Generated on Tuesday 6th April, 2010, 14:42

1. Greedy Algorithms (brief review)
   - Scheduling
   - Huffman Codes
2. Divide and Conquer (Example)
3. Dynamic Programming
4. Backtracking Algorithms
5. Another Puzzler

Nonpreemptive Scheduling

Input

A set of jobs with a running time for each

Nonpreemptive Scheduling

Input

A set of jobs with a running time for each

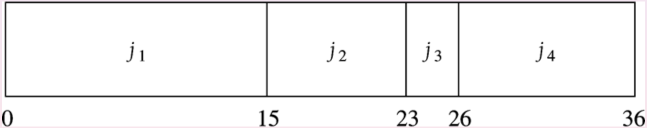
Desired output

A sequence for the jobs to execute on on single machine, minimizing the average completion time

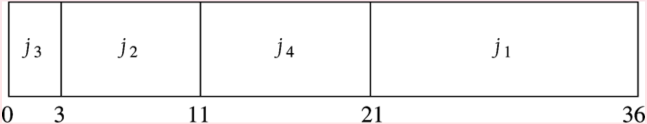
Example

Job	Time
j_1	15
j_2	8
j_3	3
j_4	10

Some schedule:



The optimal schedule:

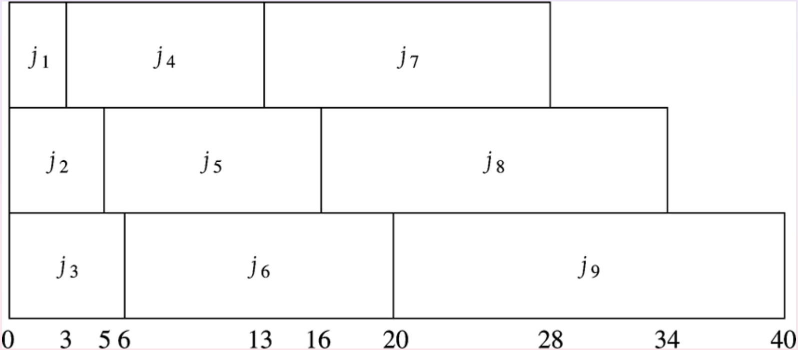


The Multiprocessor Case

N processors

Now we can run the jobs on N identical machines. What is a schedule that minimizes the average completion time?

Example



A “Slight” Variant

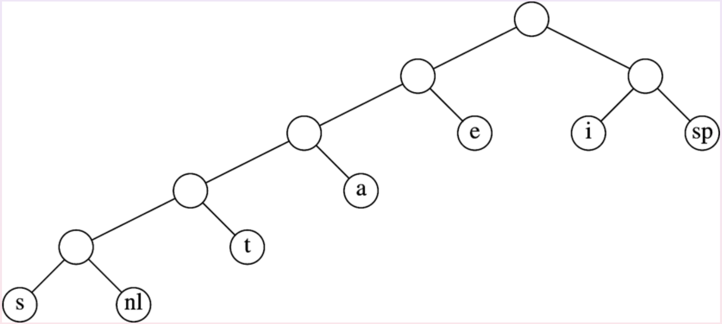
Miniming *final* completion time

If we want to minimize the *final* completion time (completion time of the last task), the problem becomes NP-complete!

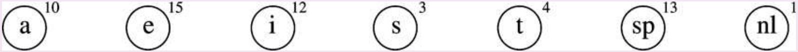
Optimal Prefix Code in Table Form

Character	Code	Frequency	Total Bits
a	001	10	30
e	01	15	30
i	10	12	24
s	00000	3	15
t	0001	4	16
space	11	13	26
newline	00001	1	5
Total			146

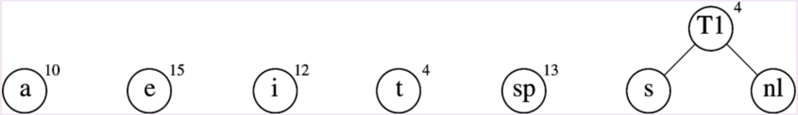
Optimal Prefix Code in Tree Form



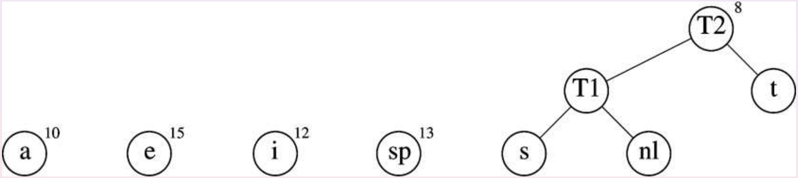
Huffman's Algorithm: An Example



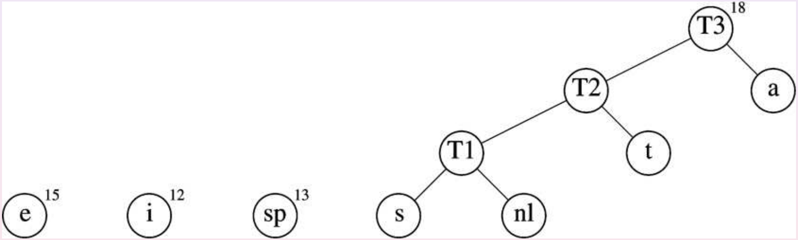
Huffman's Algorithm: An Example



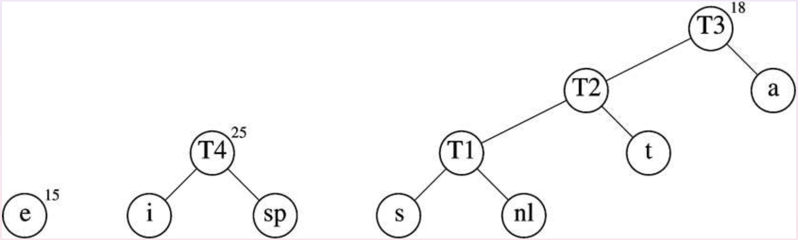
Huffman's Algorithm: An Example



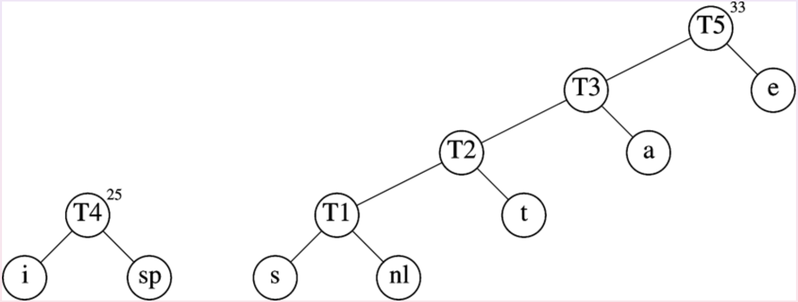
Huffman's Algorithm: An Example



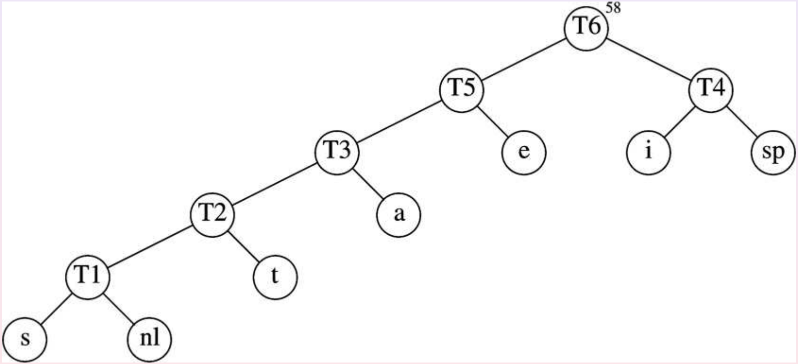
Huffman's Algorithm: An Example



Huffman's Algorithm: An Example



Huffman's Algorithm: An Example



Correctness of Huffman’s Algorithm

Observation 1

An optimal tree must be full; no node has only one child.

Correctness of Huffman's Algorithm

Observation 1

An optimal tree must be full; no node has only one child.

Observation 2

The two least frequent characters α and β must be the two deepest nodes.

Correctness of Huffman's Algorithm

Observation 1

An optimal tree must be full; no node has only one child.

Observation 2

The two least frequent characters α and β must be the two deepest nodes.

Observation 3

Characters at the same level can be swapped without affecting optimality.

Correctness of Huffman's Algorithm

Observation 3

Characters at the same level can be swapped without affecting optimality.

Correctness of Huffman's Algorithm

Observation 3

Characters at the same level can be swapped without affecting optimality.

Initial Step of Huffman's Algorithm

An optimal tree can be found that contains the two least frequent symbols as siblings; the first step in Huffman's algorithm is not a mistake.

Correctness of Huffman's Algorithm

Observation 3

Characters at the same level can be swapped without affecting optimality.

Initial Step of Huffman's Algorithm

An optimal tree can be found that contains the two least frequent symbols as siblings; the first step in Huffman's algorithm is not a mistake.

Observation 4

Every step of Huffman's algorithm produces a simplified problem, resulting from treating two characters as indistinguishable.

Correctness of Huffman's Algorithm

Observation 3

Characters at the same level can be swapped without affecting optimality.

Initial Step of Huffman's Algorithm

An optimal tree can be found that contains the two least frequent symbols as siblings; the first step in Huffman's algorithm is not a mistake.

Observation 4

Every step of Huffman's algorithm produces a simplified problem, resulting from treating two characters as indistinguishable.

Closest-Points Problem

Input

Set of points in a plane

Closest-Points Problem

Input

Set of points in a plane

Euclidean distance between p_1 and p_2

$$[(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2}$$

Closest-Points Problem

Input

Set of points in a plane

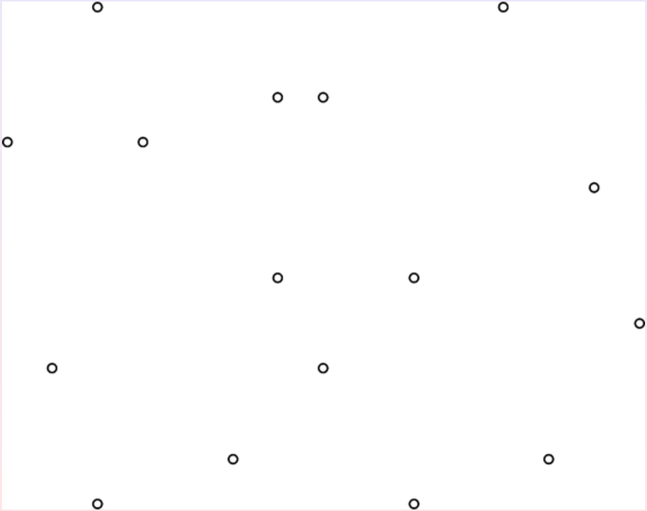
Euclidean distance between p_1 and p_2

$$[(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2}$$

Required output

Find the closest pair of points

Example



Naive Algorithm

Exhaustive search

Compute the distance between each two points and keep the smallest

Naive Algorithm

Exhaustive search

Compute the distance between each two points and keep the smallest

Run time

There are N^2 pairs to check, thus $O(N^2)$

Idea

Preparation

Sort points by x coordinate;

Idea

Preparation

Sort points by x coordinate; $O(N \log N)$

Idea

Preparation

Sort points by x coordinate; $O(N \log N)$

Divide and Conquer

Split point set into two halves, P_L and P_R.

Idea

Preparation

Sort points by x coordinate; $O(N \log N)$

Divide and Conquer

Split point set into two halves, P_L and P_R.
Recursively find the smallest distance in each half.

Idea

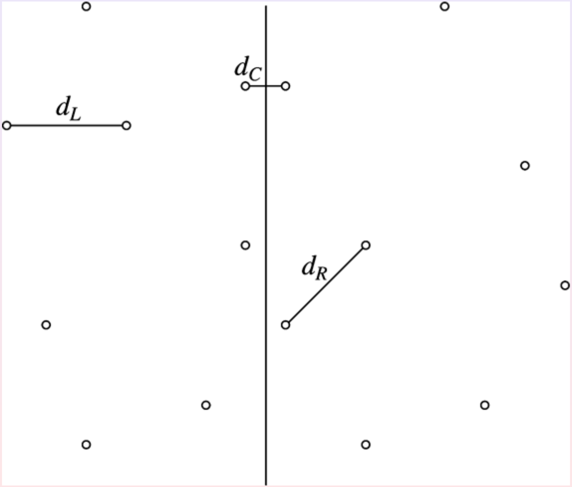
Preparation

Sort points by x coordinate; $O(N \log N)$

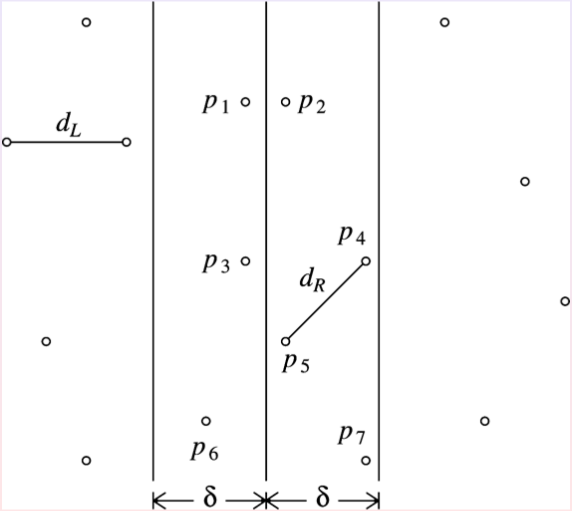
Divide and Conquer

Split point set into two halves, P_L and P_R.
Recursively find the smallest distance in each half.
Find the smallest distance of pairs that *cross* the separation line.

Partitioning with Shortest Distances Shown



Two-lane Strip



Brute Force Calculation of $min(\delta, d_C)$

```
// Points are all in the strip

for( i = 0; i < numPointsInStrip; i++ )
    for( j = i + 1; j < numPointsInStrip; j++ )
        if( dist(p_i, p_j) < δ )
            δ = dist(p_i, p_j);
```

Better Idea

Sort points by y coordinate

This allows a *scan* of the strip.

Better Idea

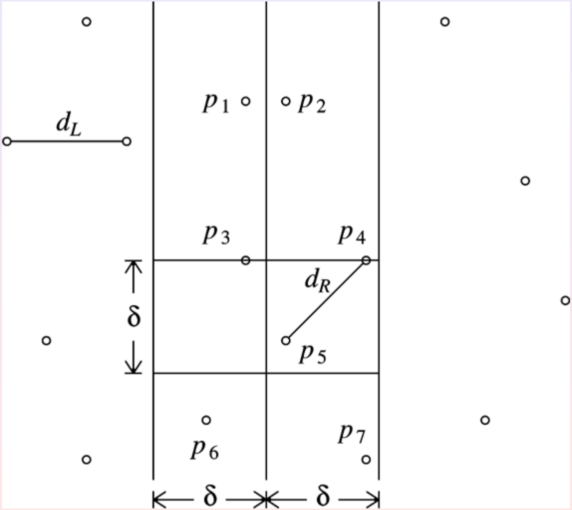
Sort points by y coordinate

This allows a *scan* of the strip.

Sort points by y coordinate

This allows a *scan* of the strip.

Only p_4 and p_5 Need To Be Considered

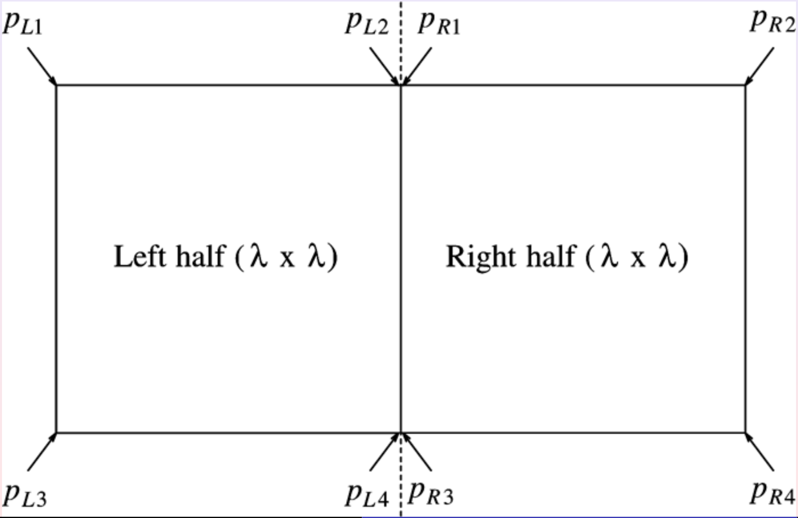


Refined Calculation of $min(\delta, d_C)$

```
// Points are all in the strip and sorted by y-coordinate

for( i = 0; i < numPointsInStrip; i++ )
    for( j = i + 1; j < numPointsInStrip; j++ )
        if( p_i and p_j's y-coordinates differ by more than δ )
            break;          // Go to next p_i.
        else
        if( dist(p_i, p_j) < δ )
            δ = dist(p_i, p_j);
```

At Most Eight Points Fit in Rectangle

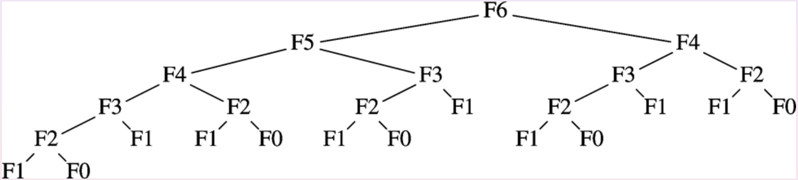


1. Greedy Algorithms (brief review)
2. Divide and Conquer (Example)
3. Dynamic Programming
   - Fibonacci Numbers
   - Optimal Binary Search Tree
   - All-pairs Shortest Path
4. Backtracking Algorithms
5. Another Puzzler

Inefficient Algorithm

```
public static int fib(int n) {
  if (n <= 1)
    return 1;
  else
    return fib(n - 1) + fib(n - 2);
}
```

Trace of Recursion



Memoization

```
int[] fibs = new int[100];
public static int fib(int n) {
  if (fibs[n]!=0) return fibs[n];
  if (n <= 1) return 1;
  int new_fib = fib(n - 1) + fib(n - 2);
  fibs[n] = new_fib;
  return new_fib;
}
```

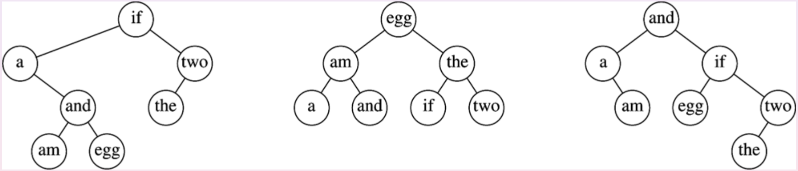
A Simple Loop for Fibonacci Numbers

```
public static int fib(int n) {
  if (n <= 1) return 1;
  int last = 1, nextToLast = 1; answer = 1;
  for (int i = 2; i <= n; i++) {
    answer = last + nextToLast;
    nextToLast = last;
    last = answer;
  }
  return answer;
}
```

Sample Input

Word	Probability
a	0.22
am	0.18
and	0.20
egg	0.05
if	0.25
the	0.02
two	0.08

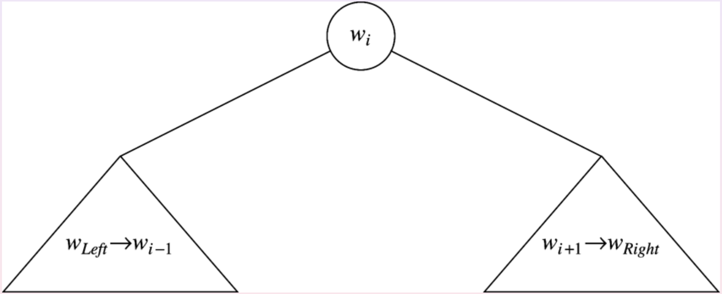
Three Possible Binary Search Trees



Comparison of the Three Trees

Input		Tree #1		Tree #2		Tree #3	
Word w_i	Probability p_i	Access Cost Once	Access Cost Sequence	Access Cost Once	Access Cost Sequence	Access Cost Once	Access Cost Sequence
a	0.22	2	0.44	3	0.66	2	0.44
am	0.18	4	0.72	2	0.36	3	0.54
and	0.20	3	0.60	3	0.60	1	0.20
egg	0.05	4	0.20	1	0.05	3	0.15
if	0.25	1	0.25	3	0.75	2	0.50
the	0.02	3	0.06	2	0.04	4	0.08
two	0.08	2	0.16	3	0.24	3	0.24
Totals	1.00		2.43		2.70		2.15

Structure of Optimal Binary Search Tree



Idea

Proceed in order of growing tree size

For each range of words, compute optimal tree

Idea

Proceed in order of growing tree size

For each range of words, compute optimal tree

Memoization

For each range, store optimal tree for later retrieval

Computation of Optimal Binary Search Tree

	Left=1		Left=2		Left=3		Left=4		Left=5		Left=6		Left=7	
Iteration=1	a..a		am..am		and..and		egg..egg		if..if		the..the		two..two	
	.22	a	.18	am	.20	and	.05	egg	.25	if	.02	the	.08	two
Iteration=2	a..am		am..and		and..egg		egg..if		if..the		the..two			
	.58	a	.56	and	.30	and	.35	if	.29	if	.12	two		
Iteration=3	a..and		am..egg		and..if		egg..the		if..two					
	1.02	am	.66	and	.80	if	.39	if	.47	if				
Iteration=4	a..egg		am..if		and..the		egg..two							
	1.17	am	1.21	and	.84	if	.57	if						
Iteration=5	a..if		am..the		and..two									
	1.83	and	1.27	and	1.02	if								
Iteration=6	a..the		am..two											
	1.89	and	1.53	and										
Iteration=7	a..two													
	2.15	and												

Run Time

For each cell of table

Consider all possible roots

Run Time

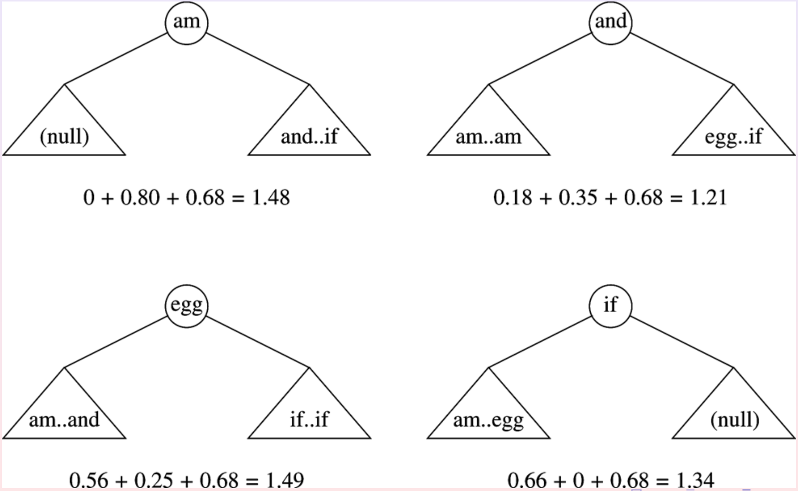
For each cell of table

Consider all possible roots

Overall runtime

$$O(N^3)$$

Example


am
(null)
and..if
0 + 0.80 + 0.68 = 1.48
and
am..am
egg..if
0.18 + 0.35 + 0.68 = 1.21
egg
am..and
if..if
0.56 + 0.25 + 0.68 = 1.49
if
am..egg
(null)
0.66 + 0 + 0.68 = 1.34

1. Greedy Algorithms (brief review)
2. Divide and Conquer (Example)
3. Dynamic Programming
4. **Backtracking Algorithms**
5. Another Puzzler

Example: The Turnpike Reconstruction Problem

From points to distances

It is easy to calculate for a set of points on a line all distances between points.

Example: The Turnpike Reconstruction Problem

From points to distances

It is easy to calculate for a set of points on a line all distances between points.

From distances to points

Consider the reverse: given the set of all distances between the points, compute the positions of the points!

Example: The Turnpike Reconstruction Problem

From points to distances

It is easy to calculate for a set of points on a line all distances between points.

From distances to points

Consider the reverse: given the set of all distances between the points, compute the positions of the points!

Placement of first point

By default, the first point is placed at position 0.

Example

Input set for 6 points

$$D = \{1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 10\}$$

Example

Input set for 6 points

$$D = \{1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 10\}$$

First and last point

First point goes to 0, which means that the last point goes to 10.

Example

Input set for 6 points

$$D = \{1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 10\}$$

First and last point

First point goes to 0, which means that the last point goes to 10.

Placement of third point

The third point can be placed either at position 2 or position 8.

Example

Input set for 6 points

$$D = \{1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 10\}$$

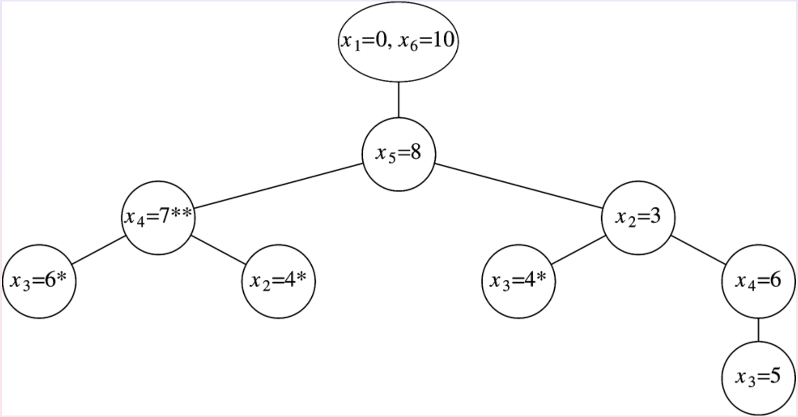
First and last point

First point goes to 0, which means that the last point goes to 10.

Placement of third point

The third point can be placed either at position 2 or position 8. This choice is arbitrary; we can flip a solution to get from one to the other!

Decision Tree for the Example



Summary of Backtracking

Decision tree

Backtracking explores decision tree; typically exponential in size

Summary of Backtracking

Decision tree

Backtracking explores decision tree; typically exponential in size

Pruning

At each node, reasoning is applied to avoid exploring the subtree

Remember Lecture 2 A: Parameter Passing

Java uses pass-by-value parameter passing.

```
public static void tryChanging(int a) {
    a = 1;
    return;
}
...
int b = 2;
tryChanging(b);
System.out.println(b);
```

Remember Lecture 2 A: Parameter Passing with Objects

```
public static void tryChanging(SomeObject obj) {
    obj.someField = 1;
    obj = new SomeObject();
    obj.someField = 2;
    return;
}
...
SomeObject someObj = new SomeObject();
tryChanging(someObj);
System.out.println(someObj.someField);
```

Remember Lecture 7 A: Sorting

Input

Unsorted array of elements

Behavior

Rearrange elements of array such that the smallest appears first, followed by the second smallest etc, finally followed by the largest element

Will This Work?

```
public static <AnyType extends Comparable<? super A
  void mergeSort( AnyType [] a) {
    AnyType[] ret = ....; // declare helper array
    .... // here goes a program that places
    .... // the element of "a" into "ret" so
    .... // that "ret" is sorted
    a = ret;
    return;
}
...
Integer[] myArray = ...;
IterativeMergeSort.mergeSort(myArray);
```

Will This Work?

```
public static <AnyType extends Comparable<? super A
  void mergeSort( AnyType [] a) {
    AnyType[] ret = ....; // declare helper array
    .... // here goes a program that places
    .... // the element of "a" into "ret" so
    .... // that "ret" is sorted
    a = ret;
    return;
}
...
Integer[] myArray = ...;
IterativeMergeSort.mergeSort(myArray);
```

Answer: No! The assignment a = ret; has no effect on myArray!

Next Lecture

Friday:

- Search trees with external storage (Section 4.7)
- Preview: Extendible hashing (Section 5.7)
- Preview: External sorting (Section 7.10)