Algorithms

MIDTERM EXAM: Thursday March 9

Lectures 1 to 10

Huffman Encoding

Each quantized value is encoded in binary.

However, in general some quantized values are more frequent than other.

Consider a simple example with four quantized values represented by the symbols A, B, C and D whose frequencies are:

$$A: 75 \text{ million}, \quad B: 2 \text{ million}, \quad C: 22 \text{ million} \quad \text{and} \quad D: 31 \text{ million}.$$

One option would be to encode each value using two bits.

How many bits does this require? *260 million*

Can we do better by encoding the values with codes of different lengths?

Yes! Here is a better encoding. $A: 0$, $B: 100$, $C: 101$ and $D: 11$.

How many bits does this require? *209 million*

Huffman Encoding

We have four quantized values A, B, C and D with frequencies:

$$A : 75 \text{ million}, \ B : 2 \text{ million}, \ C : 22 \text{ million} \ \text{ and } D : 31 \text{ million}.$$

The encoding $A : 0$, $B : 100$, $C : 101$ and $D : 11$ requires 209 million bits.

Can we do even better? How about the following encoding?

$$A : 0, \ B : 01, \ C : 11 \text{ and } D : 1.$$

This encoding is not good because it is not uniquely decipherable.

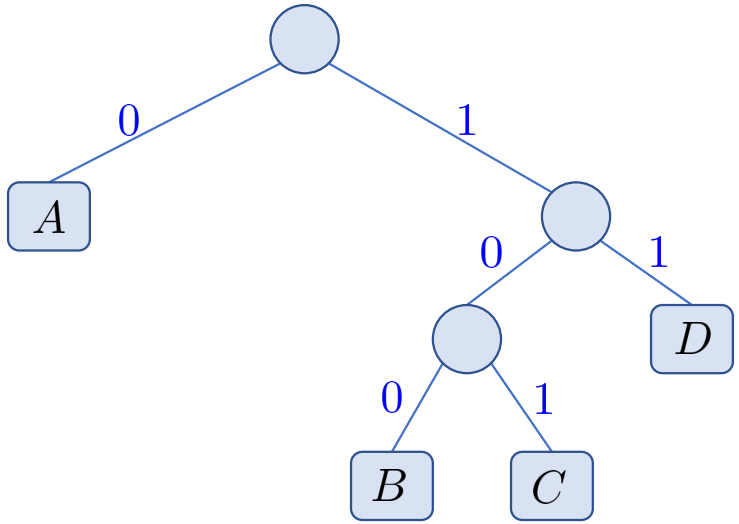
For example 11 can either be C or DD.

We will use a ***prefix-free code*** to avoid this. *No code is a prefix of another.*

Huffman Encoding

The encoding $A : 0$, $B : 100$, $C : 101$ and $D : 11$ is prefix free.

Any encoding that is prefix free can be represented as a binary tree whose leaves are the symbols.



The length of the encoding of any symbol is equal to its depth in the tree.

Let $f(A), f(B), f(C)$ and $f(D)$ denote the frequencies of A, B, C and D.

Then the length of the encoding is: $1 \cdot f(A) + 3 \cdot f(B) + 3 \cdot f(C) + 2 \cdot f(D)$.

Huffman Encoding

In general, there are n symbols whose frequences are given in an array $f[1..n]$.

We want a tree representing a prefix free codes that minimizes

$$\sum_{i=1}^{n} depth(i) \cdot f(i).$$

Huffman's algorithm builds the tree ***greedily*** as follows:

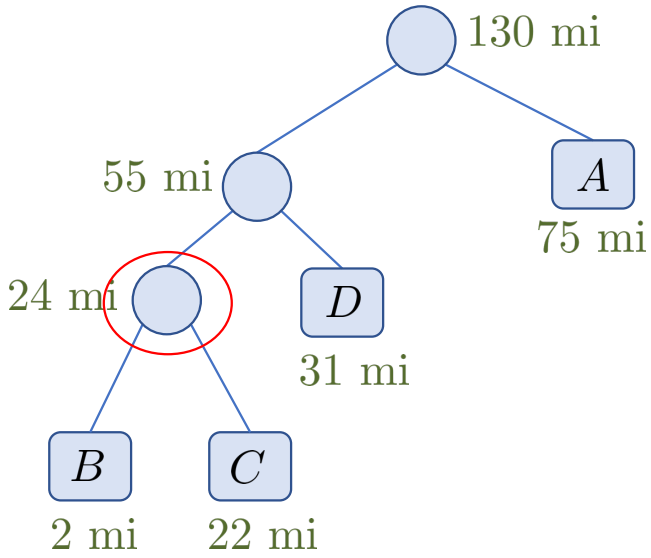
> Take the two least frequent symbols and builds a three node tree with these two symbols as the leaves. *Break ties arbitrarily.*
>
> The root of this tree represents a new combined symbol whose frequency is the sum of the frequencies of the two symbols.
>
> We now have one less symbol. **Recurse!**

Huffman Encoding

Huffman's algorithm builds the tree ***greedily*** as follows:

> Take the two least frequent symbols and builds a three node tree with these two symbols as the leaves. *Break ties arbitrarily.*
>
> The root of this tree represents a new combined symbol whose frequency is the sum of the frequencies of the two symbols.
>
> We now have one less symbol. **Recurse!**

Example. A : 75 million,
~~B : 2 million,~~
~~C : 22 million,~~
D : 31 million.



Huffman Encoding

Example.

```
THISSENTENCECONTAINSTHREEASTHREECSTWODSTWENTYSIXESFIVEFST
HREEGSEIGHTHSTHIRTEENISTWOLSSIXTEENNSNINEOSSIXRSTWENTYSEV
ENSSTWENTYTWOTSTWOUSFIVEVSEIGHTWSFOURXSFIVEYSANDONLYONEZ
```

Spaces are ignored for simplicity. In general a blank space is another symbol.

Frequency of characters:

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

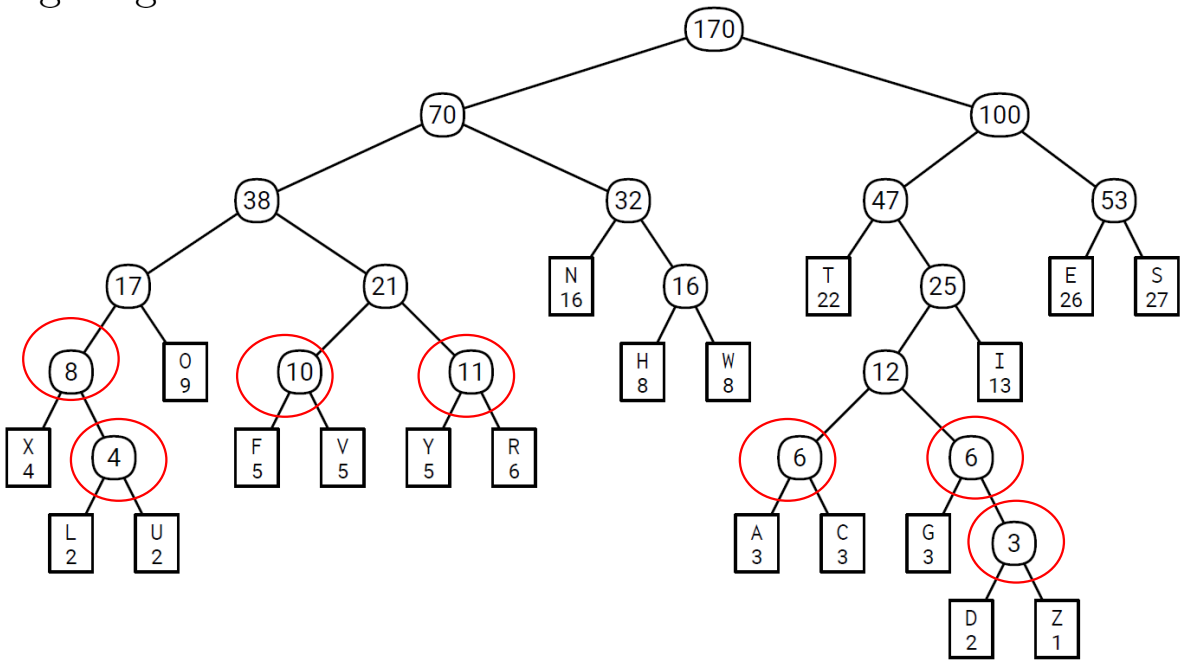
Huffman Encoding

Example.

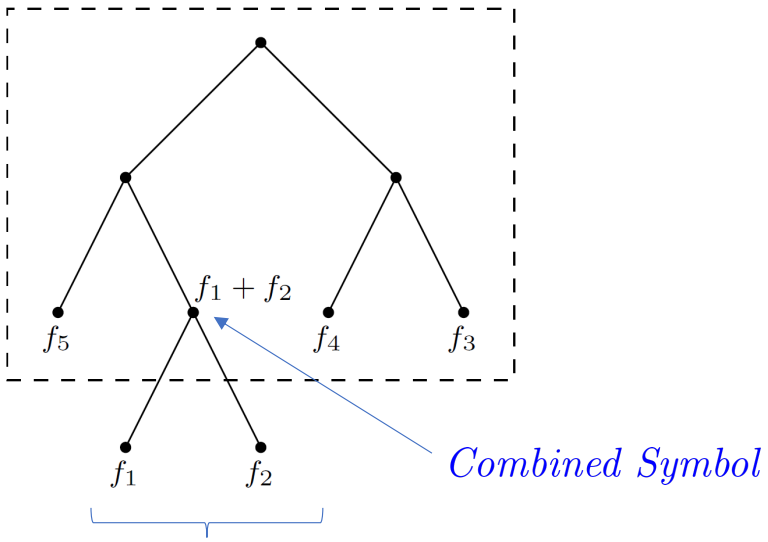
A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

Tree obtained from Huffman's algorithm:

Encoding length: 649 bits.



Huffman Encoding



How much time does it take to construct the tree according to Huffman's algorithm?

$n = \#$ *symbols*

Use priority queues

Recall priority queues

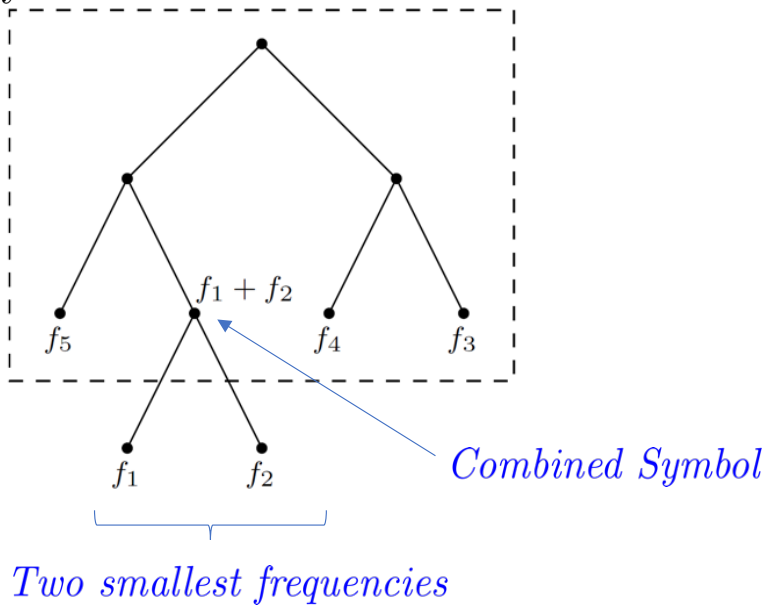
$Insert(x, p)$: insert the element x with priority p

$ExtractMin()$: remove and return the element x with the smallest priority

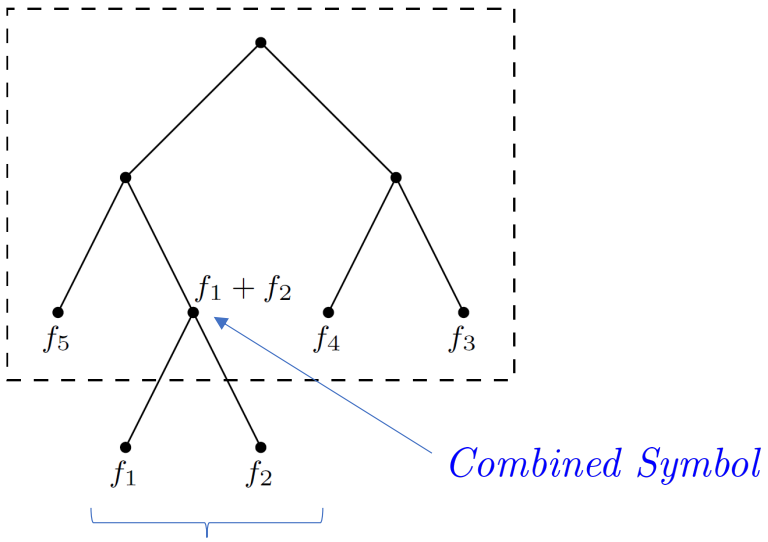
$DecreaseKey(x, \Delta)$: decrease the priority of the element x by $\Delta > 0$.

$Empty()$: returns whether the priority queue is empty.

In a simple implementation using binary heaps each of these operations takes $O(\log n)$ time where n is the number of items in the data structure.



Huffman Encoding



How much time does it take to construct the tree according to Huffman's algorithm?

Exercise: check why $O(n \log n)$ time using ***priority queues***. $n = \#$ *symbols*

Why does the algorithm work?

Huffman Encoding

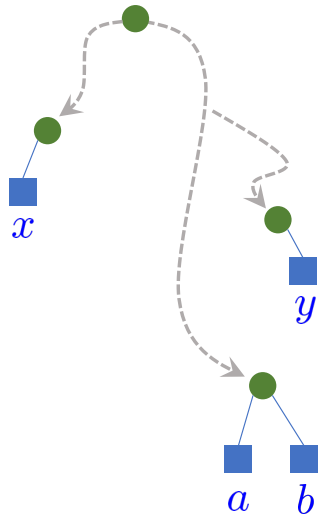
Claim. Let x and y be the two least frequent symbols (breaking ties arbitrarily). Then, there is an optimal encoding where x and y are *siblings* and have the maximum depth among all leaves.

Proof. Consider the optimal tree T. It must be a ***full binary tree*** i.e., each node is either a leaf or has two children. *Why?*

Consider the two sibling leaves a and b with the maximum depth.

Since x and y are the two least frequent symbols, exchanging these leaves with x and y does not make the encoding worse.

The claim follows. ■



This justifies Huffman's algorithm since we can combine the two least frequent symbols into a combined symbol represented by their common parent whose frequency is the sum of their frequencies, and recurse.

Algorithmic Techniques

We discuss three basic and broad algorithmic techniques in this course.

- Greedy — *Interval Scheduling, Dijkstra's Algorithm for shortest paths, Prim's and Kruskal's algorithms for MST, Huffman encoding*
- Divide and conquer
- Dynamic Programming

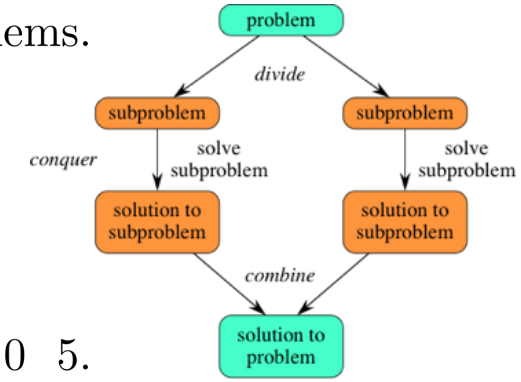
We will start with the "divide and conquer" now.

Divide and Conquer

Idea. Decompose the problem into easier subproblems.

Solve the subproblems.

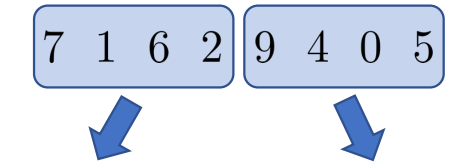
Combine the solutions to the subproblems
to obtain a solution to the original problem.



Example. Sorting an array. e.g. 7 1 6 2 9 4 0 5.

We want to split the array into two halves,
sort them recursively and combine.

Option 1.



Split into two halves. Easy!

1 2 6 7 0 4 5 9 *Sort recursively.*

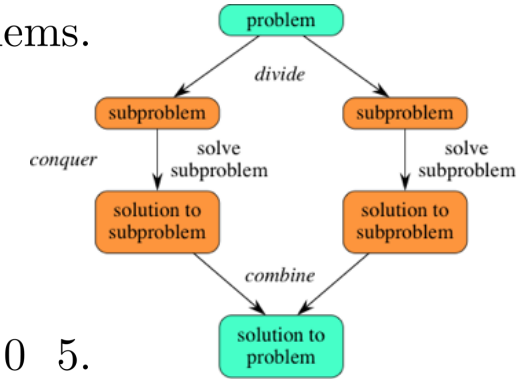
0 1 2 4 5 6 7 9 *Combine. Requires work!*

Divide and Conquer

Idea. Decompose the problem into easier subproblems.

Solve the subproblems.

Combine the solutions to the subproblems
to obtain a solution to the original problem.



Example. Sorting an array. e.g. 7 1 6 2 9 4 0 5.

We want to split the array into two halves,
sort them recursively and combine.

Option 2.

7 1 6 2 9 4 0 5

1 2 4 0 7 6 9 5

0 1 2 4 5 6 7 9

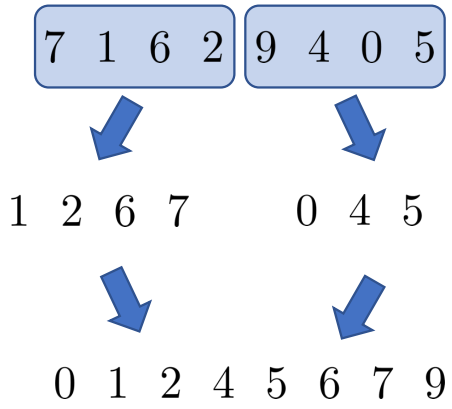
Split s.t. the no.s in the first subproblem are smaller than all the no.s in the second subproblem. Non-trivial!

Sort recursively.

Combine by concatenating. Easy!

Mergesort

Option 1.



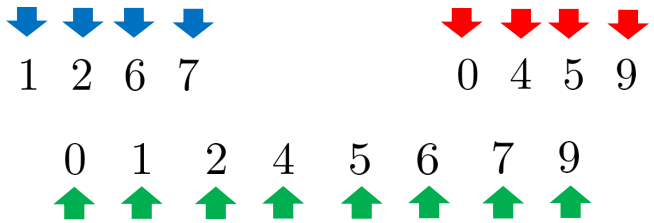
Split into two haves. Easy!

Sort recursively.

Combine. Requires work!

Non-trivial task. Merging two sorted arrays into a single sorted array.

Example:



If the total size of the two arrays is m, how many steps does this take?

$O(m)$ since we spend constant amount time per output element.

Exercise: check correctness

MERGESORT

Pseudocode.

```
MergeSort(A, p, r) {  sort A[p..r]
  if (p < r) {        ≥ 2 elements to sort
    m = (p + r)/2     index of the middle element (integer division)
    MergeSort(A, p, m)      recursively sort the left half A[p..m]
    MergeSort(A, m+1, r)    recursively sort the right half A[m+1..r]
    merge(A, p, m, r)       merge the two halves
  }
}
```

Initial call: `MergeSort(A, 0, n-1)`

Pseudocode for Merge Sort

Time to sort: T(n)

1. O(1)
2. $2T(\lceil \frac{n}{2} \rceil)$
3. cn

Merge sort A[1,..,n]

1. If n=1, done
2. Recursively sort $A[1, . . , \lceil \frac{n}{2} \rceil]$ and $A[\lceil \frac{n}{2} \rceil + 1, \dots , n]$
3. Merge the 2 sorted arrays

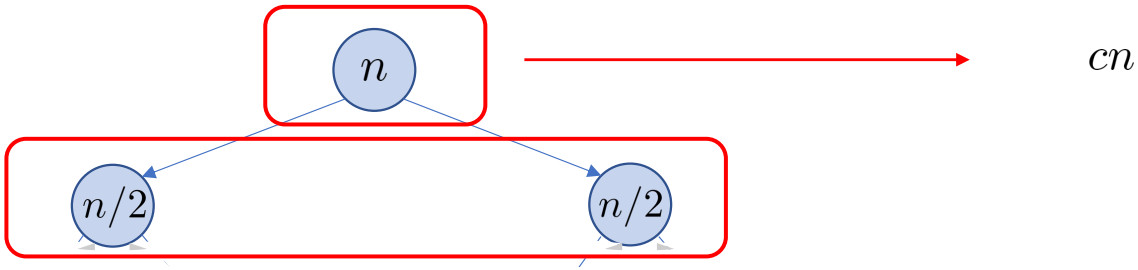
$T(n)=2T(\lceil \frac{n}{2} \rceil)+cn$

Assume n is a power of 2, $n=2^k$

Mergesort

Recurrence relation. $T(n) = 2\,T(n/2) + cn$, $T(1) = c$.

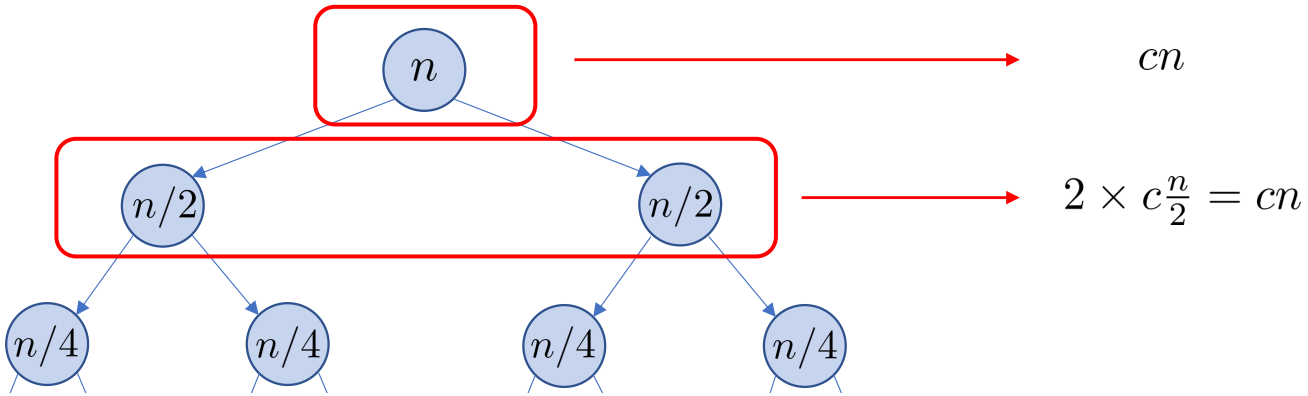
Recursion Tree



Mergesort

Recurrence relation. $T(n) = 2\,T(n/2) + cn,\ T(1) = c.$

Recursion Tree

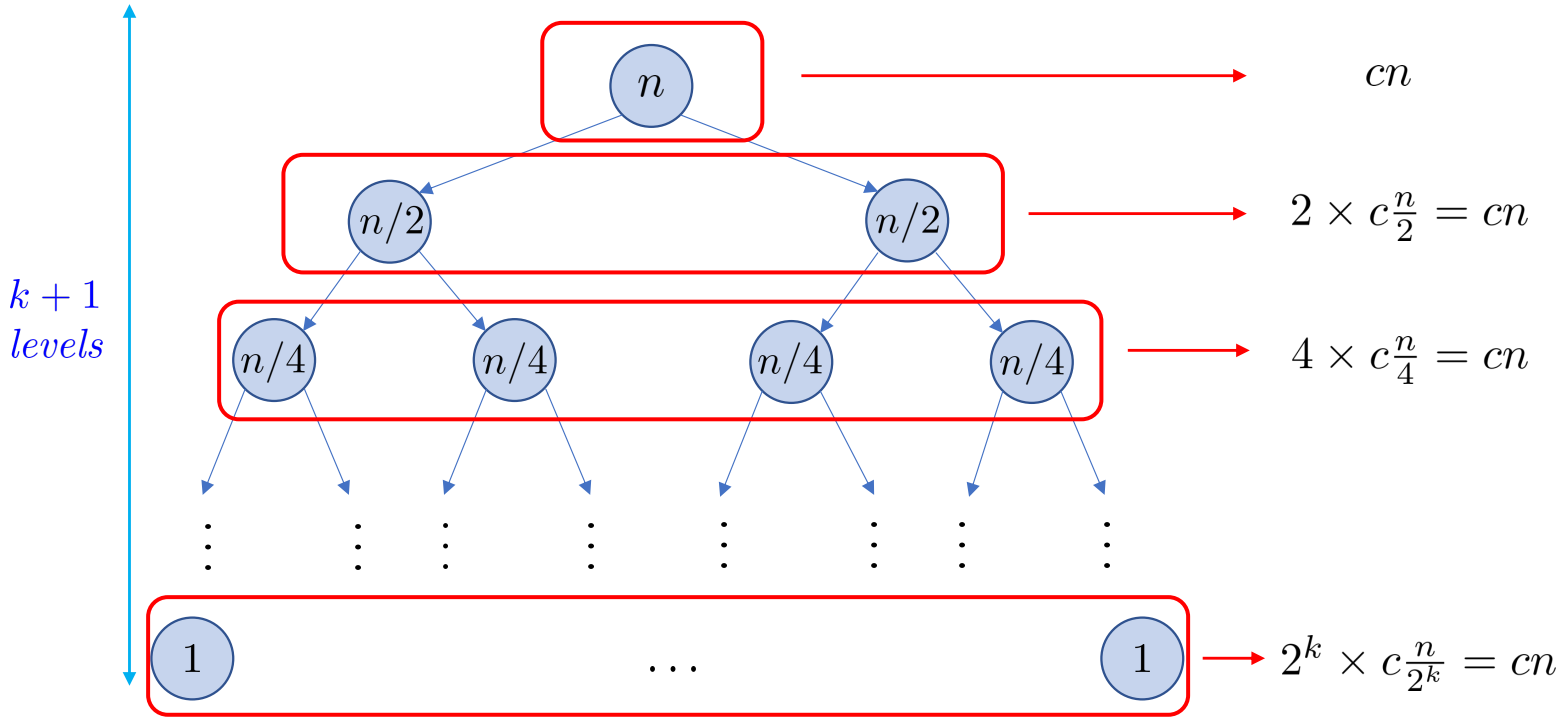


Mergesort

Recurrence relation. $T(n) = 2\,T(n/2) + cn,\ T(1) = c.$

What is the height of the tree?	**How many leaf nodes**
a) n	a) n $\log_2$ (n)
b) Independent of n	b) 1
c) 1+$\log_2$ (n)	c) $\log_2$ (n)
d) n $\log_2$ (n)	d) n

Recursion Tree



Thus $T(n) = (k+1) \cdot cn = O(n \log n)$.

Mergesort

The time complexity of Mergesort is described by the following recurrence relation:

$$T(n) = \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)}_{\textit{recursion}} + \underbrace{cn}_{\textit{merging}}$$

c : some constant

$$T(1) = c \qquad \textit{base case}$$

Solve recurrence relation using trees

For now, let us assume that $n = 2^k$ for some k so that we get rid of the floors and ceilings.

Recurrence relation. $T(n) = 2\,T(n/2) + cn$, $T(1) = c$.

Recursion tree (Homework)

1. What if n is not a power of 2?

Exercise: check correctness

MERGESORT

```
merge(A, p, m, r) {                          merge A[p..m] with A[m+1..r]

    create a new array B[0..r-p]             temporary array for storing the merged output

    i = p; j = m+1; k = 0;

    while (i <= m and j <= r) {

        if (A[i] <= A[j]) {                  next item comes from
            B[k] = A[i]; i = i+1; k = k+1;   the left subarray
        }
        else {                               next item comes from
            B[k] = A[j]; j = j+1; k = k+1;   the right subarray
        }
    }

    while (i <= m) { B[k] = A[i]; i = i+1; k = k+1; }
                                             copy any leftover
    while (j <= r){ B[k] = A[j]; j = j+1; k = k+1; }   elements

    for (k = 0 to r-p){ A[p+k] = B[k]; }     copy output back to A
}
```