

## Announcements

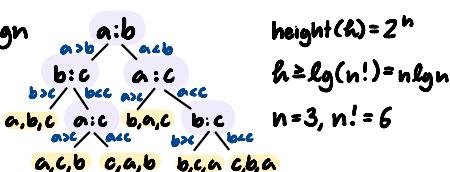
### Programming assignment #2 $\Rightarrow$ Due tomorrow

- Only use one additional array for `heap-sort()`
- Steve is here on thursday till 7:00pm for last minute questions

Many sorting algorithms are  $n \lg n$

- A decision tree that sorts  $n$  elements has height at least  $n \lg n$
- There are  $n!$  ways to sort an array of size  $n$
- The tree to the right shows why most sorts are  $n \lg n$ . They choose two elements and compares them
- If we do not compare two elements at a time, we can have a non- $n \lg n$  sort

sort  $[a, b, c]$  decision tree



$$\text{height}(k) = 2^k$$

$$k = \lg(n!) = n \lg n$$

$$n=3, n!=6$$

## Counting sort

### Assumptions

- Assume we know that all  $n$ -elements are in range  $0..k$  if  $k \leq n$  or typically  $k < n$ ,  $k \ll n$
- Then counting sort is  $O(n)$
- Practical use for  $k$  is  $\leq 8$  bits so you do not use too much space
- \* Important: Must know how to count

$k$  much less than  $n$

### Pseudo code

```
counting-sort(array A, array B, range K) {
    newarray C[K+1] = {0};
    for(i=0; i < A.length; i++) C[A[i]]++;
    for(i=1; i <= K; i++) C[i] += C[i-1];
    for(i=A.length-1; i >= 0; i++) {
        C[A[i]]--;
        B[C[A[i]]] = A[i];
    }
}
```

//A is array to sort, B is result array, K is range of ele

//init counting array

//Count number of times each element appears

//Translate counts into positions

### Example

A	0	2	3	1	3	2	2	3
C	0	0	0	0				
C	0	1	2	3				
C	1	2	3	3				
C	0	1	2	3				
C	1	2	5	8	$\Rightarrow$	0	1	2
C	0	1	2	3		0	1	2
B	0	1	2	3		0	1	2
B	0	1	2	3		3	4	5

$$K=3$$

Count number of times each element was seen

Translate to positions

Sort into new array

## Information

- Time complexity  $\Rightarrow O(n) \Rightarrow O(n+n+k)$
- Does not sort in place
- Is a stable sort, meaning in a tie, order in the original array is maintained

## Radix sort

### Radix sort $\Rightarrow$ Pseudo code

```
radix-sort(A, d) { //d is number of digits in the numbers in A
    for(i=1; i <= d; i++) { //go from most significant digit to least
        //use  $O(n)$  stable sort to sort array A elements by its digit
    }
}
```

### Bucket sort $\Rightarrow$ Pseudo code

```
bucket-sort(A, n) { //array length n
    for(i=0; i < n; i++) {
        //insert A[i] into bucket B[A[i]]
    }
    for(i=0; i < n; i++) {
        //sort bucket B using insertion sort  $O(n^2)$ 
    }
    //concatenate B[0], B[1], ... into A
}
```

} //we need enough buckets for this to be fast

### Radix-bucket sort $\Rightarrow$ Pseudo code

```
radix-bucket-sort(A, n, d) { //A: array, n: size, d: digits (9), k: range of each digit (10)
    List bucket [10];
    for(i=1; i <= d; i++) { //ith digit from lsd to msd
        for(j=0; j < n; j++) {
            r = ith digit from A[j];
            bucket[r].insertAtEnd(A[j]);
        }
        for(j=0; j < 10; j++) add items from bucket[j] into A in order
    }
}
```

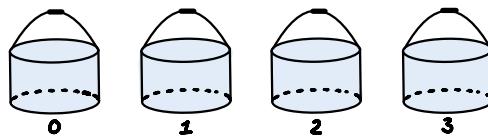
Radix-bucket sort => Example

103	202	222	102	101	213
-----	-----	-----	-----	-----	-----

101	202	222	102	103	213
-----	-----	-----	-----	-----	-----

101	202	102	103	213	222
-----	-----	-----	-----	-----	-----

101	102	103	202	213	222
-----	-----	-----	-----	-----	-----



$k=3, n=3, \text{number of buckets} = k+1 = 4$

Recombine buckets in order  
First item in bucket, first  
back in the array