Announcements

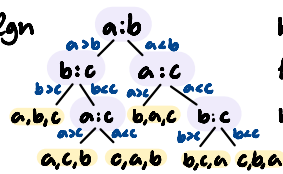
Programming assignment #2 ⇒ Due tomorrow

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

Many sorting algorythms are nlgn

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

sort [a,b,c] decision tree



height(h) = 2^h

$h \geq \lg(n!) = n\lg n$

n=3, n! = 6

Counting sort

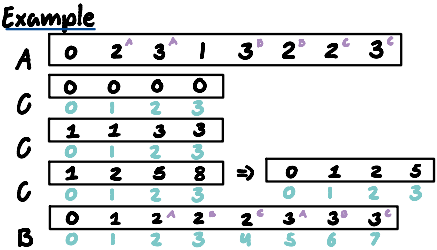
Assumptions

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

Pseudo code

```
counting_sort(array A, array B, range K) {      //A is array to sort, B is result array, K is range of ele
    new array C[K+1] = {0};                     //init counting array
    for(i=0; i<A.length; i++) C[A[i]]++;        //Count number of times each element appears
    for(i=1; i<=K; i++) C[i] += C[i-1];         //Translate counts into positions
    for(i=A.length-1; i>=0; i++) {
        C[A[i]]--;
        B[C[A[i]]] = A[i];
    }
}
```

Example



K=3

Count number of times each element was seen

Translate to positions

Sort into new array

Information

- Time complexity => O(n) => O(n+n+k)
- Does not sort in place
- Is a stable sort, meaning in a tie, order in the origional array is mantained

Radix sort

Radix sort => 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 => 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(n²)
    }
    //concationate B[0], B[1],... into A
} //we need enough buckets for this to be fast
```

Radix-bucket sort => 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[i]);
        }
        for(j=0;j<10;j++) add items from bucket[j] into A in order
    }
}
```

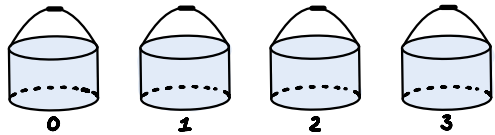
Radix-bucket sort ⇒Example

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, number of buckets = k+1 = 4

Recombine buckets in order

First item in bucket, first

back in the array