

Midterm page 1

The worst-case complexity of an algorithm is ...

a function from N (or size of instance) to N .

$\Omega(n \lg n)$ is ... a set of functions.

A random variable is ...

a function from elementary event to real number.

Polyhash is ... a set of (hash) functions.

$T(n) = 3 T(n/2) + n$ $T(n)$ is $\Theta(n^{\lg 3})$

$T(n) = 3 T(n/3) + 5$ $T(n)$ is $\Theta(n)$

$T(n) = 3 T(n/4) + n \lg n$ $T(n)$ is $\Theta(n \lg n)$

False A heap of height n can have $2^{n+1} - 1$ nodes

False A tree of height k might only have $k+1$ nodes

True A red-black can implement a priority queue efficiently

Midterm page 2

Yes, you can implement "Decrease-Key(S, x, k)" in a max heap efficiently. Go to node x (which you can do since you have a pointer to it), change its priority to k , apply Heapify (which "bubbles downward").

Given a red-black tree ... in black subtree

- Every non-leaf has 2, 3, or 4 children (by P3).
- Every leaf has depth k (by P4).
- Thus # black nodes $\geq 2^{k+1} - 1$ (obvious, or by induction)

In full red-black tree, $N \geq \# \text{ black nodes} \geq 2^{k+1} - 1$

- So $\lg N \geq \lg (2^{k+1} - 1) \geq \lg 2^k = k$; that is, $k \leq \lg N$.
- Longest path $\leq 2k$ (by P4).
- So height tree = longest path $\leq 2 \lg N$.

NOTE: Just saying "Tree is balanced" is insufficient - we never even really defined what "balanced" means.

Midterm page 3

If keys are more or less uniformly randomly distributed (so Bucket Sort take $O(N)$ time) and K is large (or grows with N , so KN isn't $O(N)$), then Bucket Sort will be faster than Radix Sort.

If the keys are very skewed, and K is small, Radix Sort will be faster (actually, all you really need is that K is very small, that is, $K \ll \lg N$).

Greedy change-making can be arbitrarily bad. Given any c , let $D_2 = 2c$, $D_3 = 2c+1$, and $N = 4c$. Then greedy will give one D_3 and $2c-1$ D_1 's, or $2c$ coins. But optimal is to give two D_2 's, or 2 coins. So greedy is c times worse.

Midterm page 5

Let $A(i)$ = best sum of numbers in the first i columns that doesn't use any entry in column i .

$B(i)$ = best sum [of ...] that uses only the top entry in column i

$C(i)$ = best sum that uses middle entry in column i

$D(i)$ = best sum that uses only the bottom entry in column i

$E(i)$ = best sum that uses top and bottom entry of column i .

I claim, given $A(i), \dots, E(i)$, we can compute $A(i+1), \dots, E(i+1)$ in the "obvious" way. This will take $\Theta(N)$ time.

E.g. $A(i+1) = \max(A(i), B(i), C(i), D(i), E(i))$

$B(i+1) = T(1,i+1) + \max(A(i), C(i), D(i)),$

etc.

Midterm, page 6

Basic idea: use divide and conquer.

```
ktiler(A,N,K) {  
  /* print the K-tiles of A[1], ..., A[N] */  
  Let M = select(A, N, K/2);      /* takes O(N) time */  
  Split A by M into Big and Small (each of size N/2);  
  if (K>1) call ktiler (Big, N/2, K/2);  
  print M;  
  if (K>1) call ktiler (Small, N/2, K/2);  
}
```

Complexity: $T(N,K) = 2T(N/2, K/2) + cN$

Recursion tree has $\lg K$ levels, each requires cN time.

So complexity is $cN \lg K$.