

## Useful facts

### Order Notation Summary:

$f(n) \in O(g(n))$  if  $\exists c > 0$  and  $n_0 \geq 0$  such that  $|f(n)| \leq c|g(n)| \forall n \geq n_0$ .

$f(n) \in \Omega(g(n))$  if  $\exists c > 0$  and  $n_0 \geq 0$  such that  $|f(n)| \geq c|g(n)| \forall n \geq n_0$ .

$f(n) \in \Theta(g(n))$  if  $\exists c_1, c_2 > 0$  and  $n_0 \geq 0$  such that  $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)| \forall n \geq n_0$ .

$f(n) \in o(g(n))$  if  $\forall c > 0 \exists n_0 \geq 0$  such that  $|f(n)| \leq c|g(n)| \forall n \geq n_0$ .

$f(n) \in \omega(g(n))$  if  $\forall c > 0 \exists n_0 \geq 0$  such that  $|f(n)| \geq c|g(n)| \forall n \geq n_0$ .

### Some useful sums:

- $\sum_{i=1}^n i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$
- $\sum_{i=1}^n \frac{1}{i} = \ln n + \gamma + o(1) \in \Theta(\log n)$
- $\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$
- $\sum_{i=0}^{\infty} \frac{i}{2^i} = 2$
- $\sum_{i=0}^n 2^i = 2^{n+1} - 1$
- $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$

### Some well-known sequences:

$n$	0	1	2	3	4	5	6	7	8	9
Power of 2, $2^n$	1	2	4	8	16	32	64	128	256	512
Factorial $n!$	1	1	2	6	24	120	720	5040	40320	362 880
Fibonacci number $F(n)$	0	1	1	2	3	5	8	13	21	34
Catalan-number $C(n)$	1	1	2	5	14	42	132	429	1430	4862

### Randomization, probability and moments:

- `random(int  $n$ )` returns an integer in  $\{0, \dots, n-1\}$ , chosen uniformly.
- $E[aX] = aE[X]$ ,  $E[X + Y] = E[X] + E[Y]$  (linearity of expectation)
- $V[X] = V[a + X]$
- Chebyshev's inequality:  $P(|X - E[X]| \geq t) \leq \frac{V(X)}{t^2}$

### Some recursions that we have seen:

Recursion	resolves to
$T(n) = T(n/2) + \Theta(1)$	$T(n) \in \Theta(\log n)$
$T(n) = 2T(n/2) + \Theta(n)$	$T(n) \in \Theta(n \log n)$
$T(n) = 2T(n/2) + \Theta(\log n)$	$T(n) \in \Theta(n)$
$T(n) = T(cn) + \Theta(n)$ for some $0 < c < 1$	$T(n) \in \Theta(n)$
$T(n) = \frac{1}{2}T(\frac{3}{4}n) + \frac{1}{2}T(n-1) + \Theta(1)$	$T(n) \in \Theta(\log n)$
$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} \max\{T(i), T(n-i-1)\} + \Theta(n)$	$T(n) \in \Theta(n)$
$T(n) = \frac{2}{n} \sum_{i=2}^{n-1} T(i) + \Theta(n)$	$T(n) \in \Theta(n \log n)$
$T(n) = \frac{4}{n} \sum_{i=2}^{n-1} T(i)$	$T(n) \in \Theta(n^3)$
$T(n) = T(\sqrt{n}) + \Theta(\sqrt{n})$	$T(n) \in \Theta(\sqrt{n})$
$T(n) = T(\sqrt{n}) + \Theta(1)$	$T(n) \in \Theta(\log \log n)$

# Some pseudocodes from the slides

## Efficient sorting with heaps

- Idea: *PQ-sort* with heaps.
- $O(1)$  auxiliary space: Use same input-array  $A$  for storing heap.

```

HeapSort(A, n)
1. // heapify
2. n ← A.size()
3. for i ← parent(last()) downto 0 do
4.   fix-down(A, i, n)

5. // repeatedly find maximum
6. while n > 1
7.   // 'delete' maximum by moving to end and decreasing n
8.   swap items at A[root()] and A[last()]
9.   decrease n
10.  fix-down(A, root(), n)
    
```

The for-loop takes  $\Theta(n)$  time and the while-loop takes  $O(n \log n)$  time.

Biedl, Kondratovsky, Petrick, Veksler (CS-UW) CS240 – Module 2 Winter 2022 22 / 25

## Making Binomial Heaps Proper

```

binomialHeap::makeProper()
1. n ← size of the binomial heap
2. compute ℓ ← ⌊log n⌋
3. B ← array of size ℓ + 1, initialized all-NIL
4. L ← list of flagged trees
5. while L is non-empty do
6.   T ← L.pop(), h ← T.height
7.   while T' ← B[h] is not NIL do
8.     if T.root.key < T'.root.key do swap T and T'
9.     // combine T with T'
10.    T'.right ← T.left, T.left ← T', T.height ← h+1
11.    B[h] ← NIL, h++
12.    B[h] ← T
13. // copy B back to list
14. for (h = 0; h ≤ ℓ; h++) do
15.   if B[h] ≠ NIL do L.append(B[h])
    
```

Biedl, Kondratovsky, Petrick, Veksler (CS-UW) CS240 – Module 2E Winter 2022 12 / 14

## Insert in Skip Lists

```

skipList::insert(k, v)
1. P ← getPredecessors(k)
2. for (i ← 0; random(2) = 1; i ← i+1) {} // random tower height
3. while i ≥ P.size() // increase skip-list height?
4.   root ← new sentinel-only list, linked in appropriately
5.   add left sentinel of root at bottom of stack P
6. p ← P.pop() // insert (k, v) in S0
7. zbelow ← new node with (k, v), inserted after p
8. while i > 0 // insert k in S1, ..., Si
9.   p ← P.pop()
10.  z ← new node with k added after p
11.  z.below ← zbelow; zbelow ← z
12.  i ← i - 1
    
```

Biedl, Kondratovsky, Petrick, Veksler (CS-UW) CS240 – Module 5 Winter 2022 9 / 17

## Merging Meldable Heaps

- Idea: Merge heap with smaller root into other one, *randomly* choose into which sub-heap to merge.
- Structural property not maintained

```

meldableHeap::merge(r1, r2)
r1, r2: roots of two heaps (possibly NIL)
returns root of merged heap
1. if r1 is NIL return r2
2. if r2 is NIL return r1
3. if r1.key < r2.key swap(r1, r2)
4. // now r1 has max-key and becomes the root.
5. randomly pick one child c of r1
6. replace subheap at c by heapMerge(c, r2)
7. return r1
    
```

Biedl, Kondratovsky, Petrick, Veksler (CS-UW) CS240 – Module 2E Winter 2022 5 / 14

## Efficient In-Place partition (Hoare)

Idea: Keep swapping the outer-most wrongly-positioned pairs.

Loop invariant:  $A \begin{matrix} \leq v & ? & \geq v & v \\ i & & j & n-1 \end{matrix}$

```

partition(A, p)
A: array of size n, p: integer s.t. 0 ≤ p < n
1. swap(A[n-1], A[p])
2. i ← -1, j ← n-1, v ← A[n-1]
3. loop
4.   do i ← i+1 while A[i] < v
5.   do j ← j-1 while j ≥ i and A[j] > v
6.   if i ≥ j then break (goto 9)
7.   else swap(A[i], A[j])
8. end loop
9. swap(A[n-1], A[i])
10. return i
    
```

Running time:  $\Theta(n)$ .

Biedl, Kondratovsky, Petrick, Veksler (CS-UW) CS240 – Module 3 Winter 2022 18 / 45

## Interpolation Search

- Code very similar to binary search, but compare at interpolated index
- Need a few extra tests to avoid crash during computation of  $m$ .

```

interpolation-search(A, n, k)
A: Sorted array of size n, k: key
1. ℓ ← 0, r ← n - 1
2. while (ℓ ≤ r)
3.   if (k < A[ℓ] or k > A[r]) return "not found"
4.   if (k = A[r]) then return "found at A[r]"
5.   m ← ℓ + ⌊  $\frac{k - A[\ell]}{A[r] - A[\ell]} \cdot (r - \ell)$  ⌋
6.   if (A[m] == k) then return "found at A[m]"
7.   else if (A[m] < k) then ℓ ← m + 1
8.   else r ← m - 1
9. // We always return from somewhere within while-loop
    
```

Biedl, Kondratovsky, Petrick, Veksler (CS-UW) CS240 – Module 6 Winter 2022 5 / 23