Asymptotic Analysis (more details)

BIG-O Notation

– Upper bound on growth of a runtime function

- $f(n) \in O(g(n))$ reads as "f(n) is big-O of g(n)"

If there exists C, n_0 such that $0 \leq f(n) \leq CG(n)$ when $n \geq n_0$ and $c > 0$.

1a. Use the definition of big-O to show $2n^2$ is big-O n^3 (find a C and n_0 that works in the above)

$$f(n) < cg(n)$$
$$2n^2 \leq cn^3 \text{ when } n > n_0 (n > 1) \text{ is } 2n^2 < cn^3$$
$$2(2^2) \leq c(2^3)$$

1b. Use the definition of big-O to show $T(n) = 3n^3 - 4n^2 + 3\lg n - n = O(n^3)$

$$0 < 3n^3 - 4n^2 + 3\lg n - n < cn^3 (\text{ find } c > 0 \text{ and } n_0 > 0 \text{ such that } n > n_0)$$
$$3n^3 - 4n^2 + 3\lg n - n ? < 3n^3$$
$$-4n^2 + 3\lg n - n ? < 0$$
$$\forall n > 0, -4n^2 \text{ is negative}$$

Note: The ? < means is this true

Omega Ω Notation – Lower Bound

$g(n) = \Omega(g(n))$
$0 \leq cg(n) \leq f(n)$
$c?$ $n_0 < n$

2. Use the definition of omega to show $n^{\frac{1}{2}} = \Omega(\log n)$

$$n > n_0, c > 0$$
$$c\lg(n) \leq n^{\frac{1}{2}}$$
$$n = 4, c\log_2(4)? \leq 4^{\frac{1}{2}}$$
$$2c \leq 2$$
$$n = 16, c\log_2(16)? \leq 16^{\frac{1}{2}}$$
$$4c \leq 4$$
$$2^{clog_2 n} \leq 2^{\sqrt{n}}$$
$$2^c 2^{log_2 n} \leq 2^{\sqrt{n}}$$
$$n2^c \leq 2^{\sqrt{n}}$$

Theta θ Notation – Strict Bound

$f(n) = \theta(g(n))$
$c_1 g(n) \leq f(n) \leq c_2 g(n)$
$c_1? < c_2?$

3a. Use the definition of theta to show $3n^3 - 4n^2 + 37n = \theta(n^3)$

$$c_1 n^3 \leq 3n^3 - 4n^2 + 37n \quad \leq c_2 n^3$$
$$2n^3? \leq 3n^3 - 4n^2 + 37n$$
$$0? \leq n^3 - 4n^2 + 37n$$
$$\text{True for } n > 0$$
$$3n^3 - 4n^2 + 37n \leq c_2 n^3$$
$$3n^3 - 4n^2 + 37n \leq 3n^3$$
$$-4n^2 + 37n \leq 0$$
$$n(-4n + 37) \leq 0 \ \forall n \geq 10$$
$$c_1 = 2, n_1 = 0, c_2 = 3, n_2 = 10$$

3b. Use the definition of theta to show $n^2 + 3n^3 = \theta(n^3)$

$$c_1 n^3 \leq n^2 + 3n^3 \qquad \leq c_2 n^3$$
$$c_1 n \leq 1 + 3n \leq \qquad c_2 n$$
$$c_1 = 2 \qquad c_2 = 4 \ \forall n > 1$$

Recursive Sorting – Mergesort

- divide and conquer (and combine) approach, recursive algorithm
- key idea: you can merge sorted lists of total length n in $\theta(n)$ linear time
- base case: a list of length one element is sorted
- For all sorting algorithms, the base case is $n = 1$.

1. Demonstrate how you can merge two sorted sub-lists total n items with n compares/copies. How much memory do we need to do this? Write pseudocode to do this.

2	3	7	8	1	4	5	6

Algorithm 1 Merge algorithm

function MERGE(A, i, j, k) ▷ i is the index of the lower sorted sequence, j is the index of the upper sorted, k is the end
  B ← array of size k
  l ← 0
  i_end ← i-1
  while i < i_end && j ≤ k **do**
    if $A[i] < A[j]$ **then**
      B[l] ← A[i]
      i++
      l++
    else
      B[l] ← A[j]
      j++
      l++
    end if
  end while
end function

Algorithm 2 Merge sort algorithm

1: **function** MERGESORT(A, p, r) ▷ Initial call Mergesort($A, 1, n$)
2:   **if** $p < r$ **then**
3:     $q \leftarrow \frac{p+r}{2}$ ▷ Integer division
4:     MergeSort(A, p, q) ▷ Recursively sort 1st half
5:     MergeSort($A, q+1, r$) ▷ Recursively sort 2nd half
6:     Merge(A, p, q, r) ▷ Merge 2 sorted sub-lists
7:   **end if**
8: **end function**

Recurrence relations for lines: $2 = c_1, 3 = c_2, 4 = 5 = T\left(\frac{n}{2}\right)$ and $O(n)$ as the runtime for merge

$$T(n) = c_1 + c_2 + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$$
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

2. Demonstrate Mergesort on this data:

	3	41	52	26	38	57	09	49
1.	3(p=1, q=4)	41	52	26	38	57	09	49(r
2.	3(p=1, q=2)	41	52	26(r=4)				
3.	3(p=1, q=1)	41 (r=2)						

I don't even know how I can write this part in LaTeX