Asymptotic Formulas:

- $O(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that
  $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$
- $\Omega(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that
  $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$
- $\Theta(g(n)) = \{f(n)$: there exists positive constants c_1, c_2 and n_0 such that
  $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$
- $o(g(n)) = \{f(n)$: for all positive constants c there exists a c constant $n_0 > 0$ such that
  $0 \leq f(n) < cg(n)$ for all $n \geq n_0\}$
- $\omega(g(n)) = \{f(n)$: for all positive constants c there exists a c constant $n_0 > 0$ such that
  $0 \leq cg(n) < f(n)$ for all $n \geq n_0\}$
- You may use these[8], or any other functions we discussed in class in your answers:
  - sorting: INSERTION−SORT(A), MERGE−SORT(A, p, r), QUICK−SORT(A, p, r), HEAPSORT(A)
  - searching: RECURSIVE−BINARY−SEARCH(A, v, low, high)
  - red-black tree operations: RB-INSERT(T, x), RB-SEARCH(T, k), RB-DELETE(T, x)
  - dynamic order statistics: OS-SELECT(T, i), OS-RANK(T, x)
  - interval trees: INTERVAL−SEARCH(T, i)
  - max-priority queue operations: INSERT(S,x), MAXIMUM(S), EXTRACT−MAX(S), INCREASE−KEY(S,x,k)
  - min-priority queue operations: INSERT(S,x), EXTRACT-MIN(S), DECREASE−KEY(S,x,k), MINIMUM(S)
  - Heap operations: MIN−HEAPIFY(A,i), BUILD−MAX−HEAP(A), MIN−HEAP−INSERT(A,i), HEAPSORT(A), BUILD−MIN−HEAP(A), MAX−HEAPIFY(A,i), MAX−HEAP−INSERT(A,i), HEAP-EXTRACT-MAX(A), HEAP-EXTRACT-MIN(A)
  - Linked List Operations: LIST-SEARCH(L,k), LIST-INSERT(L,x), LIST-DELETE(L,x)
    Linked List attributes: L.head, L.tail, L.key
  - Stack Operations: PUSH(S,x), POP(S)
  - Queue Operations: ENQUEUE(Q,x), DEQUEUE(Q)
  - Order Statistics: SELECT(A,i), RANDOMIZED-SELECT(A,i) or RANDOMIZED-SELECT(A,p,r,i)
  - Hashing operations: CHAINED−HASH−INSERT(T,x), CHAINED−HASH−SEARCH(T, k), CHAINED−HASH−DELETE(T,x)
  - Matrix Multiplication: Strassen's algorithm which runs in time $T(n) = 7T(n/2) + \Theta(n^2)$. Please note that $T(n)$ is $\Theta(n^{\log_2 7})$

The (simplified) Master Method
$a \geq 1$, $b > 1$ and $c \geq 0$:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ aT(n/b) + \Theta(n^c) & \text{otherwise} \end{cases}$$

1. if $\log_b a > c$ then $T(n) = \Theta(n^{\log_b a})$
2. if $\log_b a = c$ then $T(n) = \Theta(n^c \log(n))$
3. if $\log_b a < c$ then $T(n) = \Theta(n^c)$

The Master Method $a \geq 1$, $b \geq 2$ and $f(n) \geq 0$:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ aT(n/b) + f(n) & \text{otherwise} \end{cases}$$

1. if $f(n)$ is $O(n^{log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$
2. if $f(n)$ is $\Theta(n^{log_b a} \log^k n)$ constant $k \geq 0$
   then $T(n) = \Theta(n^{\log_b a} \log^{k+1}(n))$
3. if $f(n)$ is $\Omega(n^{log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$

Markov's Inequality: For X, a discrete random variable, $Pr[X \geq t] \leq \frac{E[X]}{t}$

[8]When A is a parameter, the book occasionally includes its size, n. I have not included, n, in the parameter list You may include it when you call the function.

Binary Heap Functions

```
MAX-HEAPIFY(A, i)
  l = LEFT(i)
  r = RIGHT(i)
  if l <= A.heap-size and A[l] > A[i]
     largest = l
  else largest = i
  if r <= A.heap-size and A[r] > A[largest]
     largest = r
  if largest not equal to i
     exchange A[i] with A[largest]
     MAX-HEAPIFY(A,largest)
```

```
HEAP-EXTRACT-MAX(A)
 if A.heap-size < 1
   error ‘‘heap underflow’’
 max = A[1]
 A[1] = A[A.heap-size]
 A.heap-size = A.heap-size -1
 MAX-HEAPIFY(A, 1)
 return max
```

```
HEAP-INCREASE-KEY(A, i, key)
 if key<A[i]
    error ‘‘new key is smaller than current key’’
 A[i] = key
 while i > 1 and A[PARENT(i) < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

```
MAX-HEAP-INSERT(A, key)
 A.heap-size = A.heap-size + 1
 A[A.heap-size] = minus infinity
 HEAP-INCREASE-KEY(A, A.heap-size, key)
```

The Merge-sort Algorithm

```
MERGE-SORT(A, p, r)
  if p < r
    q =  floor((p + r)/2 )
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q+1, r)
    MERGE(A, p, q, r)
```

Strassen’s Matrix Multiplication Algorithm $AB = C$

$P_1 = A_{11}(B_{12} - B_{22})$
$P_2 = (A_{11} + A_{12})B_{22}$
$P_3 = (A_{21} + A_{22})B_{11}$
$P_4 = A_{22}(B_{21} - B_{11})$
$P_5 = (A_{11} + A_{22})(B_{11} + B_{22})$
$P_6 = (A_{12} - A_{22})(B_{21} + B_{22})$
$P_7 = (A_{11} - A_{21})(B_{11} + B_{12})$
$C_{11} = P_5 + P_4 - P_2 + P_6$
$C_{12} = P_1 + P_2$
$C_{21} = P_3 + P_4$
$C_{22} = P_5 + P_1 - P_3 - P_7$

Red-Black Tree Functions

```
LEFT-ROTATE(T,x)
  y = x.right
  x.right = y.left
  if y.left != T.nil
      y.left.p = x
  y.p = x.p
  if x.p == T.nil
      T.root =y
  elseif x == x.p.left
       x.p.left = y
  else
       x.p.right = y
  y.left = x
```

```
RB-INSERT(T,z)
  y = T.nil
  x = T.root
  while x != T.nil
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
  z.p = y
  if y == T.nil
     T.root = z
  elseif z.key < y.key
     y.left = z
  else y.right = z
  z.left = T.nil
  z.right = T.nil
  z.color = RED
  RB-INSERT-FIXUP(T,z)
```

Tree Algorithms

```
ITERATIVE-TREE-SEARCH(T, k)
  x = T.root
  while x not equal NIL
           and k not equal to x.key
       if k < x.key
            x = x.left
       else x = x.right
  return x
```

Tree is augmented:size attribute

```
OS-RANK(T,x)
 r = x.left.size + 1
 y=x
 while y not equal T.root
      if y == y.p.right
           r = r + y.p.left.size + 1
      y = y.p
return r
```

```
OS-SELECT(x, i)
r = x.left.size + 1
if i == r
   return x
elseif i < r
   return OS-SELECT(x.left, i)
else return OS-SELECT(x.right, i - r)
```