1 Useful Information

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[1], 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 $T(n) = \Theta(f(n))$

[1] 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

```
0 MAX-HEAPIFY(A, i)
1   l = LEFT(i)
2   r = RIGHT(i)
3   if l <= A.heap-size and A[l] > A[i]
4      largest = l
5   else largest = i
6   if r <= A.heap-size and A[r] > A[largest]
7      largest = r
8   if largest not equal to i
9      exchange A[i] with A[largest]
10     MAX-HEAPIFY(A,largest)
```

```
0 HEAP-EXTRACT-MAX(A)
1  if A.heap-size < 1
2    error ``heap underflow''
3  max = A[1]
4  A[1] = A[A.heap-size]
5  A.heap-size = A.heap-size -1
6  MAX-HEAPIFY(A, 1)
7  return max
```

```
0 HEAP-INCREASE-KEY(A, i, key)
1  if key<A[i]
2     error ``new key is smaller than current key''
3  A[i] = key
4  while i > 1 and A[PARENT(i) < A[i]
5     exchange A[i] with A[PARENT(i)]
6     i = PARENT(i)
```

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

The Merge-sort Algorithm

```
0 MERGE-SORT(A, p, r)
1   if p < r
2     q =  floor((p + r)/2 )
3     MERGE-SORT(A, p, q)
4     MERGE-SORT(A, q+1, r)
5     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

```
0 LEFT-ROTATE(T,x)
1   y = x.right
2   x.right = y.left
3   if y.left != T.nil
4       y.left.p = x
5   y.p = x.p
6   if x.p == T.nil
7       T.root =y
8   elseif x == x.p.left
9        x.p.left = y
10  else
11       x.p.right = y
12  y.left = x
```

```
0 RB-INSERT(T,z)
1   y = T.nil
2   x = T.root
3   while x != T.nil
4     y = x
5     if z.key < x.key
6         x = x.left
7     else x = x.right
8   z.p = y
9   if y == T.nil
10     T.root = z
11  elseif z.key < y.key
12     y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T,z)
```

Tree Algorithms

```
0 ITERATIVE-TREE-SEARCH(T, k)
1   x = T.root
2   while x not equal NIL
              and k not equal to x.key
3        if k < x.key
4              x = x.left
5        else x = x.right
6   return x
```

Tree is augmented:size attribute

```
0 OS-RANK(T,x)
1  r = x.left.size + 1
2  y=x
3  while y not equal T.root
4       if y == y.p.right
5            r = r + y.p.left.size + 1
6       y = y.p
7 return r
```

```
0 OS-SELECT(x, i)
1 r = x.left.size + 1
2 if i == r
3    return x
4 elseif i < r
5    return OS-SELECT(x.left, i)
6 else return OS-SELECT(x.right, i - r)
```