周子龙 1851201

Answer Sheet:

1. 2, log$n$, $n^{2/3}$, 4n^2, 3^n, n !

2. (1) $f(n) = \Theta(g(n))$

   For $f(n) = logn^2 = 2logn$, thus f(n) and g(n) only differ in constant coefficient.

   (2) $f(n) = O(g(n))$

   For $f(n) = logn^2 = 2logn$ and $g(n) = n^{0.5}$, thus the former expression has lower rank class then the latter.

   (3) $f(n) = \Omega(g(n))$

   As L'hospital's law points out, $\lim_{n\to\infty}\frac{n}{ln^2 n} = \lim_{n\to\infty}\frac{n}{2\ln n} = \lim_{n\to\infty}\frac{n}{2} = \infty$, (here we substitute $log^2 n$ with $ln^2 n$, for there is only a constant factor different, hence this will not cause any different in the final answer), therefore $f(n)$ has a higher rank class than $g(n)$.

   (4) $f(n) = \Omega(g(n))$

   $f(n) = ng(n) + n$, and $g(n)$ itself has a lower rank than n.

   (5) $f(n) = \Theta(g(n))$

   $f(n), g(n)$ are both constant.

   (6) $f(n) = \Omega(g(n))$

   $f(n) = g(n)^2$, and both of them are not constant.

   (7) $f(n) = \Omega(g(n))$

   (8) $f(n) = O(g(n))$

   Both of the expressions have an exponential growth order, but f(n) have a smaller base then g(n).

3. 1) n+6

Let $T'(n) = \frac{3\times 2^n}{64} = 3\times 2^{n-6}$ be the running time of the second machine, given $t = 3 \times 2^n$, compare the two equations above, we can get $t = T'(n+6)$. Hence, using the same algorithm and in the same given time, the second machine can solve a problem set of $n+6$.

2) 8n

3) any given magnitude.

For T'(n) = 1/8, thus for any given magnitude of problem set, the second machine can solve within the given time.

4. **Algorithm** ImprovedBinarySearch(A[0…n-1], x)

//Implements of nonrecursive improved binary search

//Input: An array A[0…n-1] sorted in ascending order and a search key x.

//Output: If x is not in the array, return the biggest index i and the smallest index j, which A[i] < x < A[j]. If x is in the array, the return value i, j will be of the same value. In extreme cases where there is no bigger/smaller key value than x, algorithm will return -1.

```
i←0, j←n-1
while i < j do
    mid ← (i+j)/2
    if A[mid] < x do
        i ← mid+1
    else if A[mid] > x do
        j ← mid-1
    else
        i ← mid
        j ← mid
if A[i] < x do
    return j, -1
else if A[i] == x do
    return i, i
else do
    return i-1, i
```

The basic operation for above algorithm is comparison. For a presorted array A, the key may occur at any position, thus the possibility of finding it is $\frac{1}{n}$, where n is the length of a given array.

Hence, we can get following conclusion.

1) If the key happens to lie in the middle of the array,

$$C_{best}(n) = 1$$

2) If the key happens to be the first or the last value of the array, or even does not reside the given array, in such scenario

$$C_{worst}(n) = C_{worst}\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + 1$$

$$C_{worst}(1) = 1$$

using 2^k substitute n, we may obtain:

$$C_{worst}(2^k) = k + 1 = \log_2 n + 1$$

$$C_{worst}(n) = \lfloor\log_2 n\rfloor + 1$$

3) For average case,

$$C_{avg}(n) = \log_2 n$$

```
input:70   , A[i] = 66   , A[j] = 78
input:60   , A[i] = 51   , A[j] = 66
input:1    , A[i] = -1   , A[j] = 2
input:10000, A[i] = 6783 , A[j] = -1
input:51   , A[i] = 51   , A[j] = 51
logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
```