

CST8130: Data Structures: Quiz 1: Fall 2007

Answer all questions on the mark sense cards. Use a pencil to fill in the mark sense card (a pen will not record your answers correctly). Erase corrections thoroughly. This quiz is different than the one posted on my website.

1. What is the Big-O for the following algorithm?

```
i = 1;
while (i < n) {
    i = i + 5;
    j = n;
    while (j > 0)
        j = j - 2;
}
```

- a) $O(n^2)$
 - b) $O(2 \log_2 n)$
 - c) $O(n \log_2 n)$
 - d) $O(\frac{1}{2} \log_2 n)$
 - e) none of the above.
2. Select the sequence of Big-O that ranges from best performance to poorest performance.
- a) $O(n)$ $O(\log_2 n)$ $O(n \log_2 n)$ $O(n^2)$ $O(2^n)$
 - b) $O(\log_2 n)$ $O(n)$ $O(n \log_2 n)$ $O(n^2)$ $O(2^n)$
 - c) $O(n)$ $O(\log_2 n)$ $O(n^2)$ $O(n!)$ $O(n \log_2 n)$
 - d) $O(\log_2 n)$ $O(n)$ $O(n \log_2 n)$ $O(2^n)$ $O(n^2)$
 - e) none of the above.
3. Identify the code required to create a dynamically allocated array of pointers to *RE* (*RouterEntry*) objects given the variable *nSize*.
- a) `RE paRE** = new *RouterEntry[nSize];`
 - b) `RE** paRE = newRouterEntry*[nSize];`
 - c) `RE* paRE = newRouterEntry[nSize];`
 - d) `RE* paRE = newRouterEntry**[nSize];`
 - e) none of the above.
4. Identify the code required to create a dynamically allocated array of *RE* (*RouterEntry*) objects given the variable *nSize*.
- a) `RE paRE** = new *RouterEntry[nSize];`
 - b) `RE** paRE = newRouterEntry*[nSize];`
 - c) `RE* paRE = newRouterEntry[nSize];`
 - d) `RE* paRE = newRouterEntry**[nSize];`
 - e) none of the above.

5. What is the Big-O for the following algorithm?

```
for (int i = 0; i < n; i++) {
    for (int j = n; j > 0; j = j / 2)
        ; // do something
    }
for (int j = n; j > 0; j = j - 2)
    ; // do something else
```

- a) $O(2n^2)$
 - b) $O(n^2 \log_2 n)$
 - c) $O(n^2)$
 - d) $O(n^3)$
 - e) $O(n \log_2 n)$
6. What is the Big-O notation for an algorithm with the following efficiency?

$4n + \log_2 n + \frac{1}{2}n + 3n$

- a) $O(n)$
 - b) $O(\log_2 n)$
 - c) $O(4n \log_2 n)$
 - d) $O(\log_2 n + n)$
 - e) none of the above.
7. Given the following code, what will be output with a call to *Recur*(4,7)?

```
int Recur(int n, int m)
{
    cout << n << " ";
    if (m <= 2)
        return 1;
    else
        return n * Recur(m, n-1);
}
```

- a) 4 6 3 5
- b) 4 7 3 6 2 5
- c) 4 7 3 6
- d) Infinite recursion – blow stack
- e) none of the above.

Notice that the arguments *m* and *n* switch positions in the recursive call.