Review

1. What is the output of the following algorithm when $n = 6$, $n = 8$, and $n = 10$?
2. What is the time complexity t_n? You may assume n is divisible by 2.

```
j = 1;
while ( j <= n / 2)
     i = 1;
     while ( i <= j)
          cout << j << i;
          i++;
     j++;
```

Answer

1. What is the output of the following algorithm when $n = 6$, $n = 8$, and $n = 10$?

When $n = 6$, output is 11 21 22 31 32 33
When $n = 8$, output is 11 21 22 31 32 33 41 42 43 44

2. What is the time complexity t_n? You may assume n is divisible by 2.

$t_n = n/2 + (n/2) - 1 + \ldots + 2 + 1$

Time Complexity = $n^2 / 8 + n / 4$

$\Theta(n^2)$

Review

3. Draw the recursion tree that is created when `Quicksort` and `Mergesort` are performed on this array. Label the ordering of steps for `MergeSort`.
   { 2, 4, 11, 5, 3, 8, 10, 1, 3, 7, 15, 12 }

4. Draw a chart reflecting a top-level call to `partition` on the same array.

i	j	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	S[9]	S[10]	S[11]	S[12]
-	-	2	4	11	5	3	8	10	1	3	7	15	12
2	1	2	4	11	5	3	8	10	1	3	7	15	12
3	1	2	4	11	5	3	8	10	1	3	7	15	12
4	1	2	4	11	5	3	8	10	1	3	7	15	12
5	1	2	4	11	5	3	8	10	1	3	7	15	12
6	1	2	4	11	5	3	8	10	1	3	7	15	12
7	1	2	4	11	5	3	8	10	1	3	7	15	12
8	1 → 2	2	4	11	5	3	8	10	1	3	7	15	12
9	2	2	1	11	5	3	8	10	4	3	7	15	12
10	2	2	1	11	5	3	8	10	4	3	7	15	12
11	2	2	1	11	5	3	8	10	4	3	7	15	12
12	2	2	1	11	5	3	8	10	4	3	7	15	12
-	2	1	2	11	5	3	8	10	4	3	7	15	12

Review

5. Write a function `doubleArray` that takes an integer array of any size as a parameter. This function should double the size of the array

Answer

Note: pseudocode style of answers will vary, which is okay!

```
doubleArray(keytype A[])
{
     keytype newArr = new keytype[A.length * 2];
     for (i = 0 to A.length)
          newArr[i] = A[i];
     return newArr
}
```

Review

6. Write a recursive algorithm that searches a sorted list of n items by dividing it into three sublists of almost $n/3$ items each. This algorithm will test the element at position $n/3$ and the element at position $2n/3$. It finds the sublist that might contain the target item, and divides the list into three smaller sublists of almost equal size. It repeats this process until it finds the item or concludes that it is not in the list.
7. Analyze your algorithm by defining its recurrence relation, and give the worst-case time complexity result.

Review

```
search (index low, index high, number target)
      if low > high
            return -1;
      if low == high
            if(A[low] == target)  return low;
            else return -1;
      else
            index first_mid_point = low + ⌊(high - low) / 3⌋
            if (A[first_mid_point] == target)  return first_mid_point
            else if (target < A[mid])
                  search(low, first_mid_point - 1, target)
            else
                  index second_mid_point = 2 * first_mid_point;
                  if (A [second_mid_point] == target)  return second_mid_point;
                  else if (target < A[second_mid_point])
                        search (first_mid_point + 1, second_mid_point - 1, target);
                  else
                        search (second_mid_point + 1, high, target);
```

Review

This search algorithm cuts the input in thirds each time, recursively checking ⅓ of the input each time. This gives us the following recurrence relation:

$T_1 = 1$

$T_n = T_{n/3} + 1$

Expand:

$T_{n/3} = T_{n/3^2} + 1$

$T_{n/3^2} = T_{n/3^3} + 1$ which leads to:

$T_n = T_{n/3^3} + 1 + 1 + 1$ $\qquad$ $T_n = T_{n/3^k} + k$

We are at a base case when $n/3^k = 1$, so $n = 3^k$ or $k = \log_3 n$

$1 + \log_3 n$ is the final time complexity.