More sorting

4/24/26

Administrivia

- HW 4 (list methods and running times) due Tuesday

Where we were

- Sorting: putting values in order
  - In general, use a Comparator for the order (we're just going to sort integers)
  - Assume they're in an array
- Selection sort
  - Repeatedly find smallest value and put it in front
  - $O(n^2)$

Insertion sort

(code adapted from Wikipedia)

- Keep sorted portion of the array and grow it by inserting new values into the proper place

```
for(int i=0; i < A.length; i++) {        //add value A[i]
    int j = i;                           //iterator to seek possible places
    while((j > 0) && (A[j-1] > A[j])) {
        swap A[j] and A[j-1];
        j--;
    }
}
```

Insertion sort

(code adapted from Wikipedia)

- Keep sorted portion of the array and grow it by inserting new values into the proper place

```
for(int i=0; i < A.length; i++) {      //add value A[i]
    int j = i;                         //iterator to seek possible places
    while((j > 0) && (A[j-1] > A[j])) {
        swap A[j] and A[j-1];
        j--;
    }
}
```

What most accurately characterizes the running time of insertion sort?

A. O(1)
B. O(n)
C. $O(n^2)$
D. $O(n^3)$
E. None of the above

Insertion sort

(code adapted from Wikipedia)

- Keep sorted portion of the array and grow it by inserting new values into the proper place

```
for(int i=0; i < A.length; i++) {       //add value A[i]
    int j = i;                          //iterator to seek possible places
    while((j > 0) && (A[j-1] > A[j])) {
        swap A[j] and A[j-1];
        j--;
    }
}
```

What most accurately characterizes the running time of insertion sort?

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. $O(n^3)$
E. None of the above

Bubble sort

(code adapted from Wikipedia)

- Repeatedly find out-of-order pairs and swap them

```
boolean swapped = true;
while(swapped) {
   swapped = false;
   for(int i=0; i < A.length-1; i++)
        if(A[i] > A[i+1]) {
            swap A[i] and A[i+1];
            swapped = true;
        }
}
```

Bubble sort

(code adapted from Wikipedia)

- Repeatedly find out-of-order pairs and swap them

```
boolean swapped = true;
while(swapped) {
   swapped = false;
   for(int i=0; i < A.length-1; i++)
        if(A[i] > A[i+1]) {
             swap A[i] and A[i+1];
             swapped = true;
        }
}
```

What most accurately characterizes the running time of bubble sort?

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. $O(n^3)$
E. None of the above

Bubble sort

(code adapted from Wikipedia)

- Repeatedly find out-of-order pairs and swap them

```
boolean swapped = true;
while(swapped) {
   swapped = false;
   for(int i=0; i < A.length-1; i++)
       if(A[i] > A[i+1]) {
           swap A[i] and A[i+1];
           swapped = true;
       }
}
```

What most accurately characterizes the running time of bubble sort?

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. $O(n^3)$
E. None of the above

Which of the sorting algorithms could cause the execution shown below?

5	1	4	2	3
1	5	4	2	3
1	2	4	5	3
1	2	3	5	4

A. Selection sort
B. Insertion sort
C. Bubble sort
D. None of these
E. More than one of these

Which of the sorting algorithms could cause the execution shown below?

5	1	4	2	3
1	5	4	2	3
1	2	4	5	3
1	2	3	5	4

A. Selection sort
B. Insertion sort
C. Bubble sort
D. None of these
E. More than one of these

Which of the sorting algorithms could cause the execution shown below?

5	1	4	2	3
1	5	4	2	3
1	4	5	2	3
1	4	2	5	3

A. Selection sort
B. Insertion sort
C. Bubble sort
D. None of these
E. More than one of these

Which of the sorting algorithms could cause the execution shown below?

5	1	4	2	3
1	5	4	2	3
1	4	5	2	3
1	4	2	5	3

A. Selection sort
B. Insertion sort
C. Bubble sort
D. None of these
E. More than one of these (insertion & bubble)

Can we do better than $O(n^2)$ time?

Merge sort

- Split array into two equal-sized pieces, recursively sort each half, and merge them back

```
Mergesort(A[1..n]) {
    if(n > 1) {
        m = (n+1)/2;
        copy 1st m values of A into array L and rest into array R;
        Mergesort(L);
        Mergesort(R);
        Merge(L, R, A);
    }
}
```

```
Merge(A[1..n₁], B[1..n₂], R[1..(n₁+n₂)]) {
    //merge sorted arrays A and B into R
    int i = 1, j = 1;            //i is position in A, j is position in B
    for(int k = 1; k <= (n₁+n₂); k++) {//k is position in R
        if(i ≤ n₁ and (j > n₂ or A[i] ≤ B[j])) {
            R[k] = A[i];
            i++;
        } else {
            R[k] = B[j];
            j++;
        }
}
```

```
Merge(A[1..n_1], B[1..n_2], R[1..(n_1+n_2)]) {
    //merge sorted arrays A and B into R
    int i = 1, j = 1;            //i is position in A, j is position in B
    for(int k = 1; k <= (n_1+n_2); k++) {//k is position in R
        if(i ≤ n_1 and (j > n_2 or A[i] ≤ B[j])) {
            R[k] = A[i];
            i++;
        } else {
            R[k] = B[j];
            j++;
        }
}
```

What most accurately characterizes the running time of Merge? ($n = n_1 + n_2$)

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. None of the above

```
Merge(A[1..n₁], B[1..n₂], R[1..(n₁+n₂)]) {
    //merge sorted arrays A and B into R
    int i = 1, j = 1;          //i is position in A, j is position in B
    for(int k = 1; k <= (n₁+n₂); k++) {//k is position in R
        if(i ≤ n₁ and (j > n₂ or A[i] ≤ B[j])) {
            R[k] = A[i];
            i++;
        } else {
            R[k] = B[j];
            j++;
        }
}
```

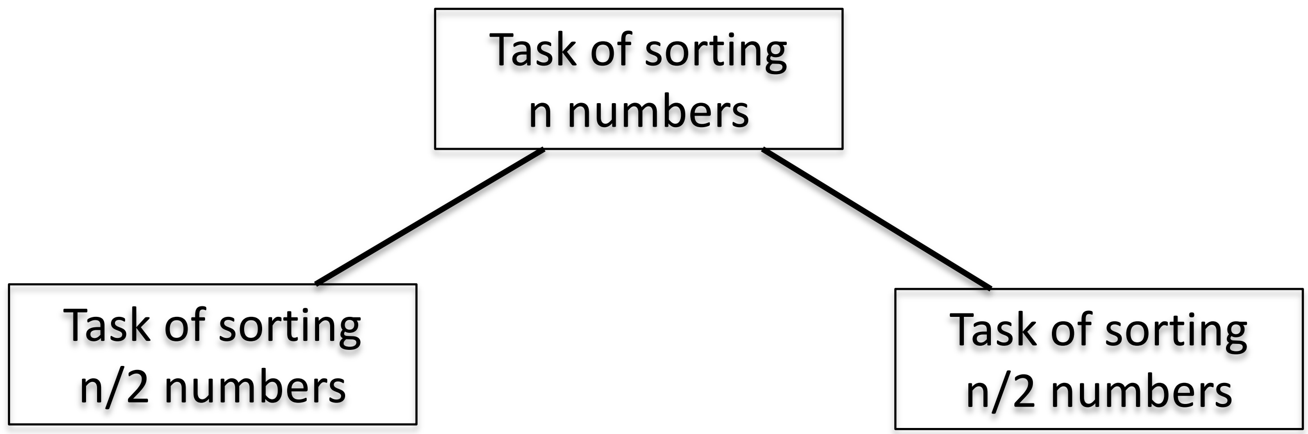
What most accurately characterizes the running time of Merge? ($n = n_1 + n_2$)

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. None of the above

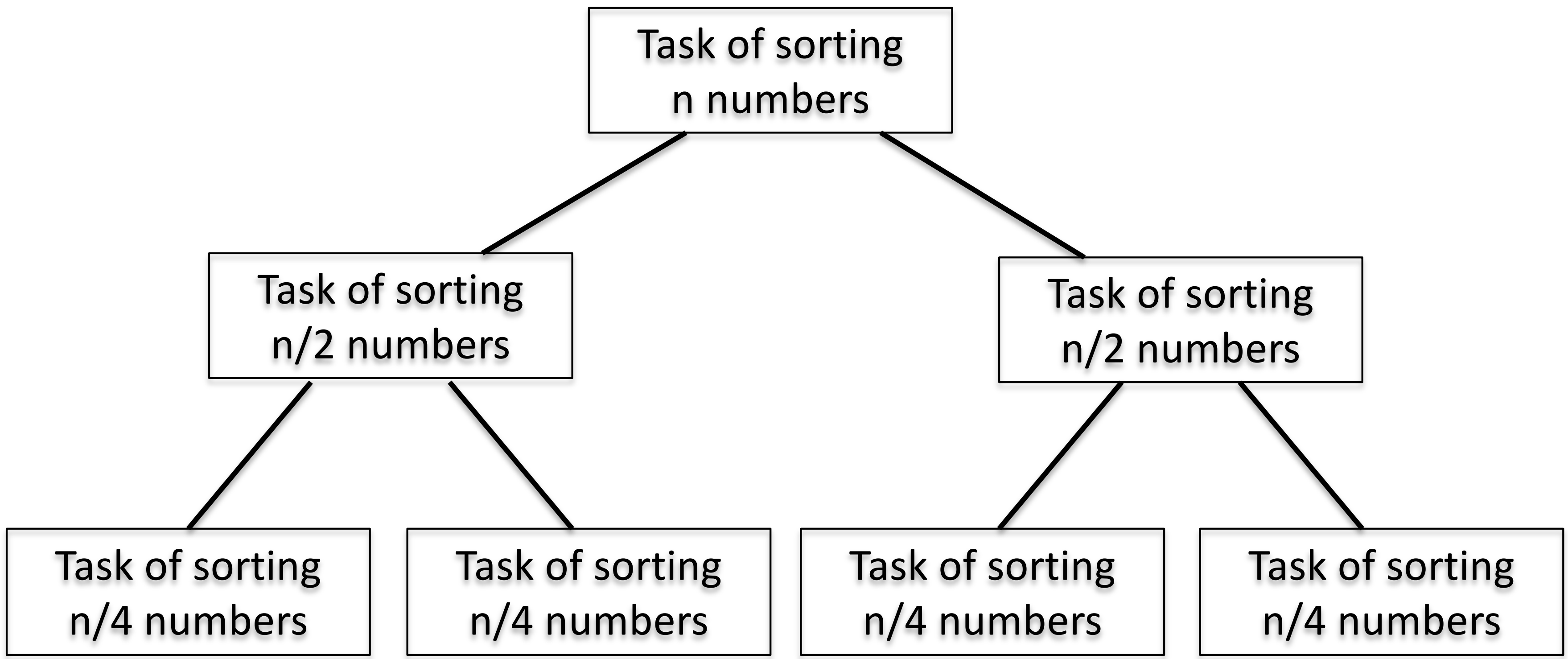
Running time of mergesort

Task of sorting
n numbers

Running time of mergesort

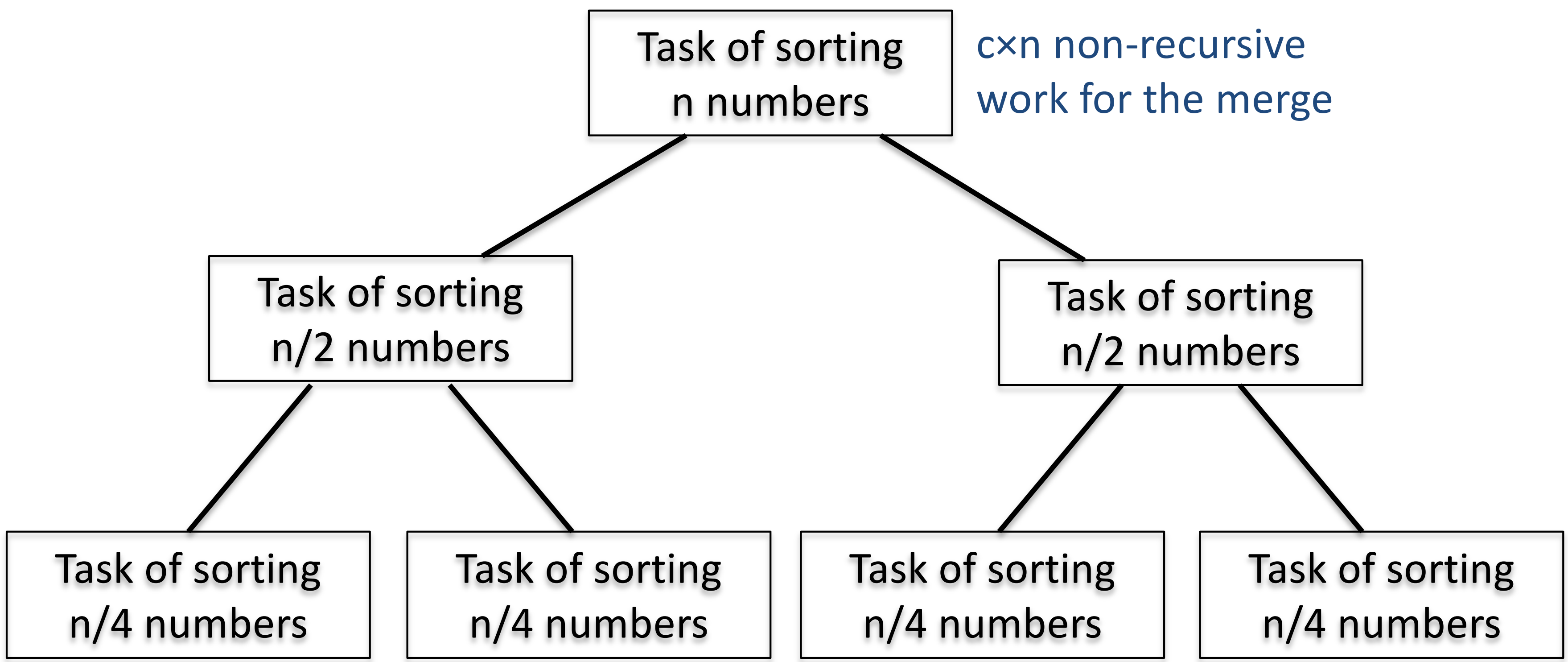


Running time of mergesort



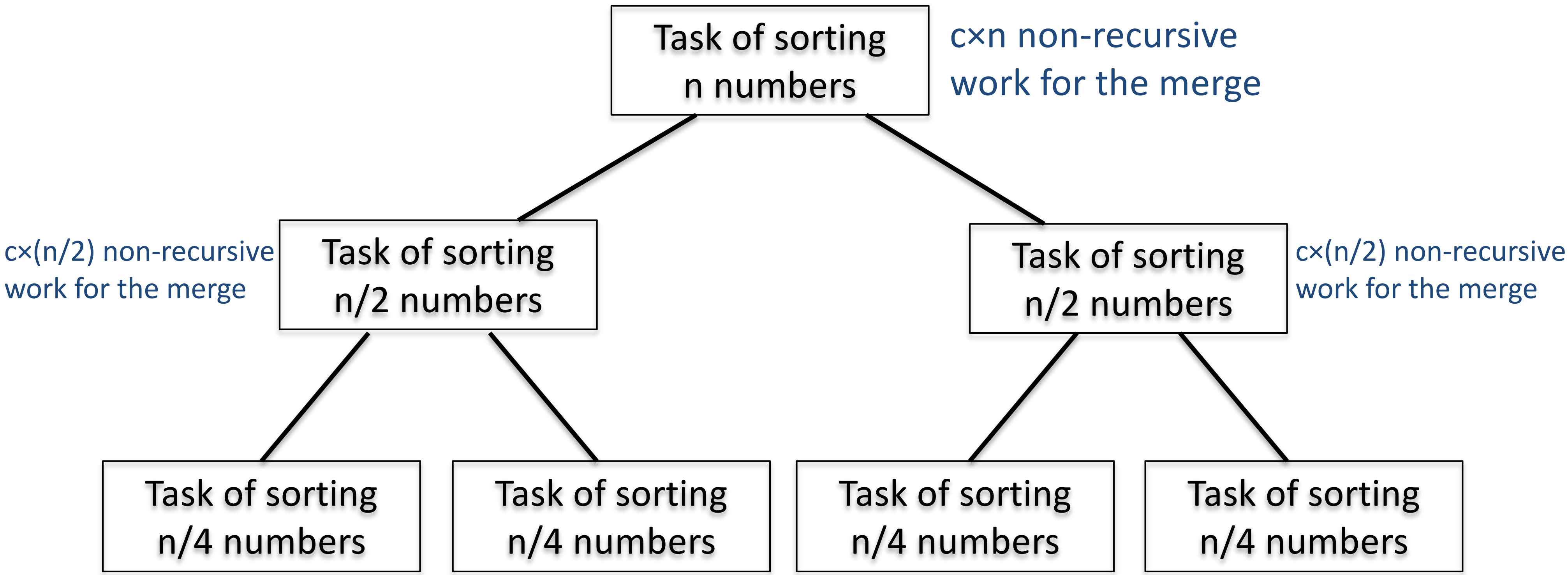
... and so on

Running time of mergesort



... and so on

Running time of mergesort



... and so on

Logarithms!

Logarithms

$\log_b x$ = "log base b of x"

= power of b that gives x

= number of times you can divide x by b before getting 1

Logarithms

$\log_b x$ = "log base b of x"

= power of b that gives x

= number of times you can divide x by b before getting 1

$2^0 = 1$ so $\log_2 1$ is 0

$2^1 = 2$ so $\log_2 2$ is 1

$2^2 = 4$ so $\log_2 4$ is 2 (and $\log_2 3$ is between 1 & 2)

What is $\log_2 32$?

A. 3
B. 4
C. 5
D. 6
E. None of the above

What is $\log_2 32$?

A. 3
B. 4
C. 5
D. 6
E. None of the above

What is $\log_2 100$?

A. Between 6 and 7
B. Between 7 and 8
C. Between 8 and 9
D. Between 9 and 10
E. None of the above

What is $\log_2 100$?

A. Between 6 and 7
B. Between 7 and 8
C. Between 8 and 9
D. Between 9 and 10
E. None of the above

What is $\log_2 1000$?

A. Between 7 and 8
B. Between 8 and 9
C. Between 9 and 10
D. Between 10 and 11
E. None of the above

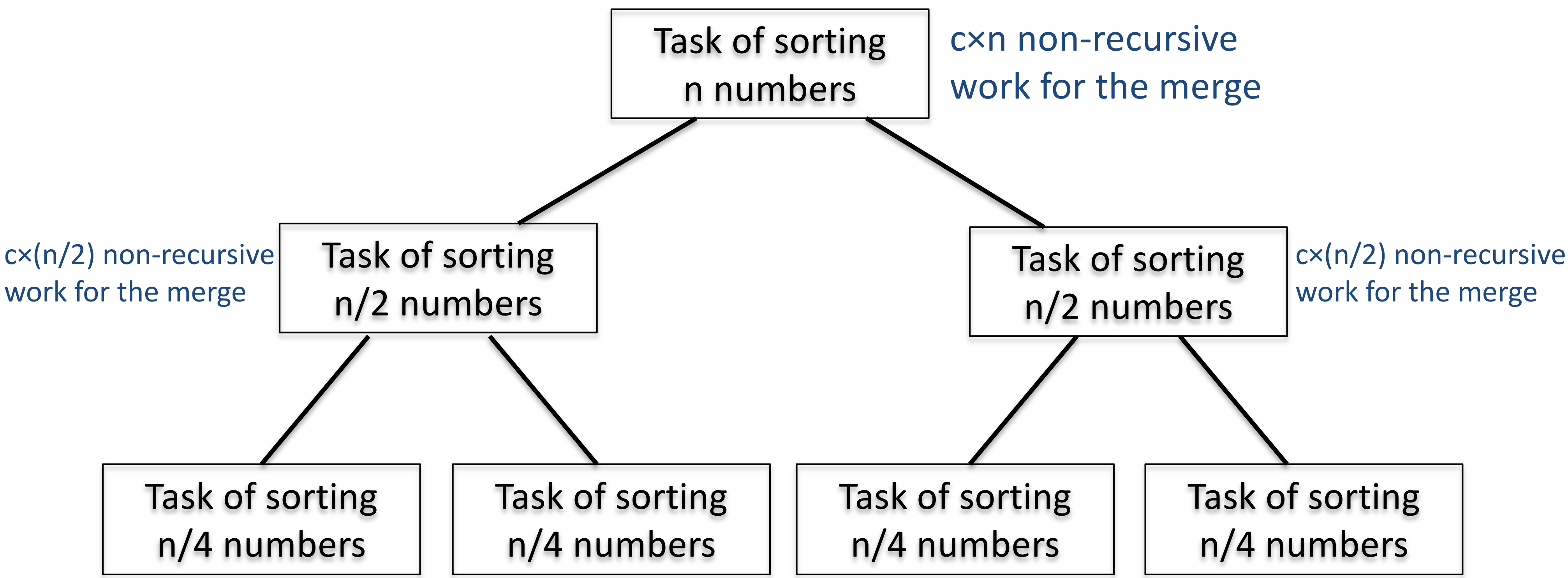
What is $\log_2 1000$?

A. Between 7 and 8
B. Between 8 and 9
C. Between 9 and 10
D. Between 10 and 11
E. None of the above

Properties of logarithms

- $\log_b xy = \log_b x + \log_b y$
- $\log_b (x/y) = \log_b x - \log_b y$
- $\log_b x^y = y \log_b x$
- $\log_b a = \dfrac{\log_c a}{\log_c b}$ for any c

Recall: Running time of mergesort



... and so on