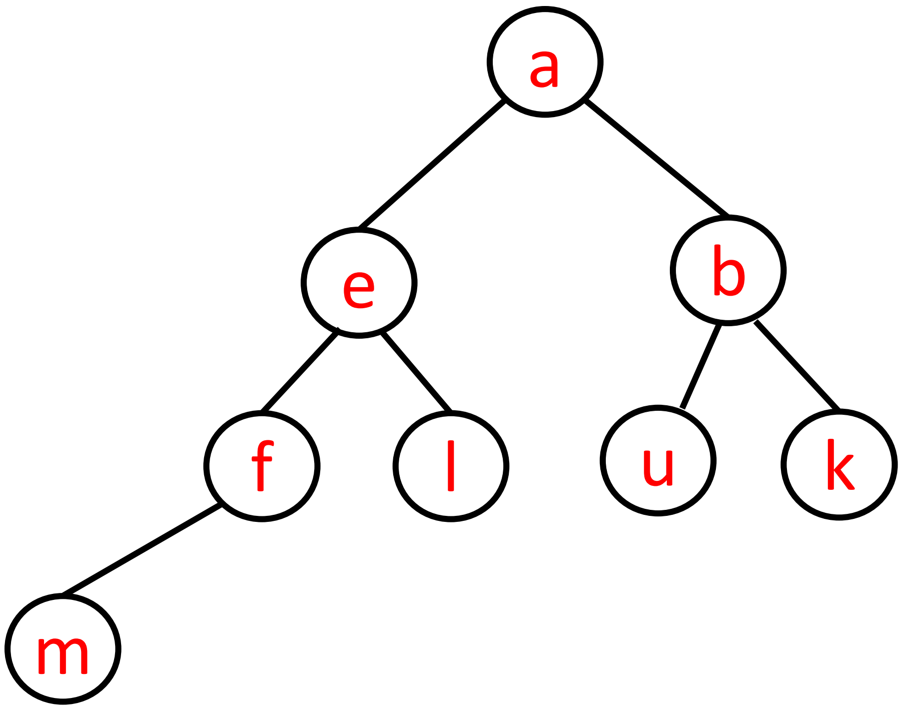
COMP 250

Lecture 23

heaps 2

Nov. 2, 2016

RECALL: min Heap (definition)



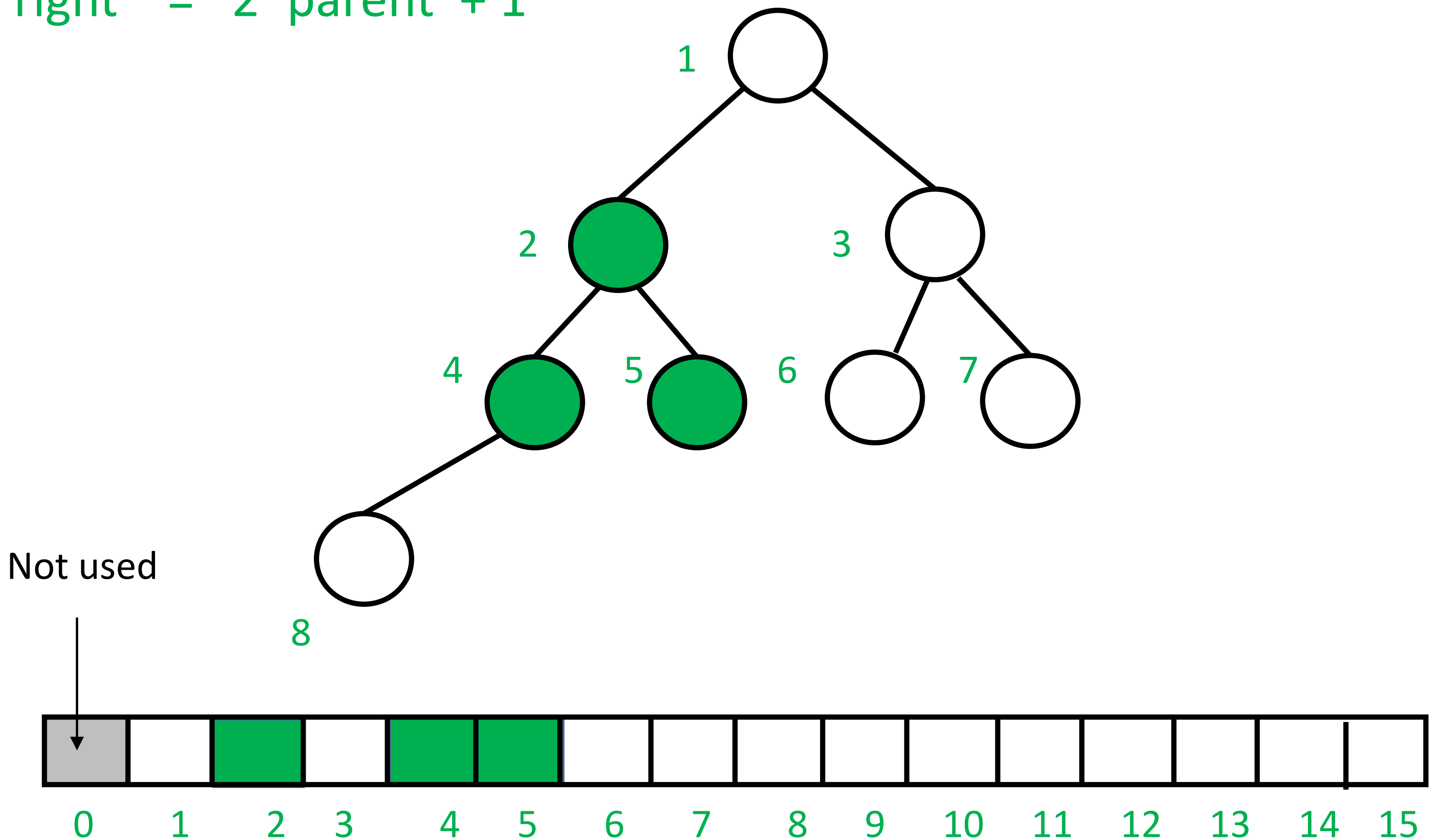
Complete binary tree with (unique) comparable elements, such that each node's element is less than its children's element(s).

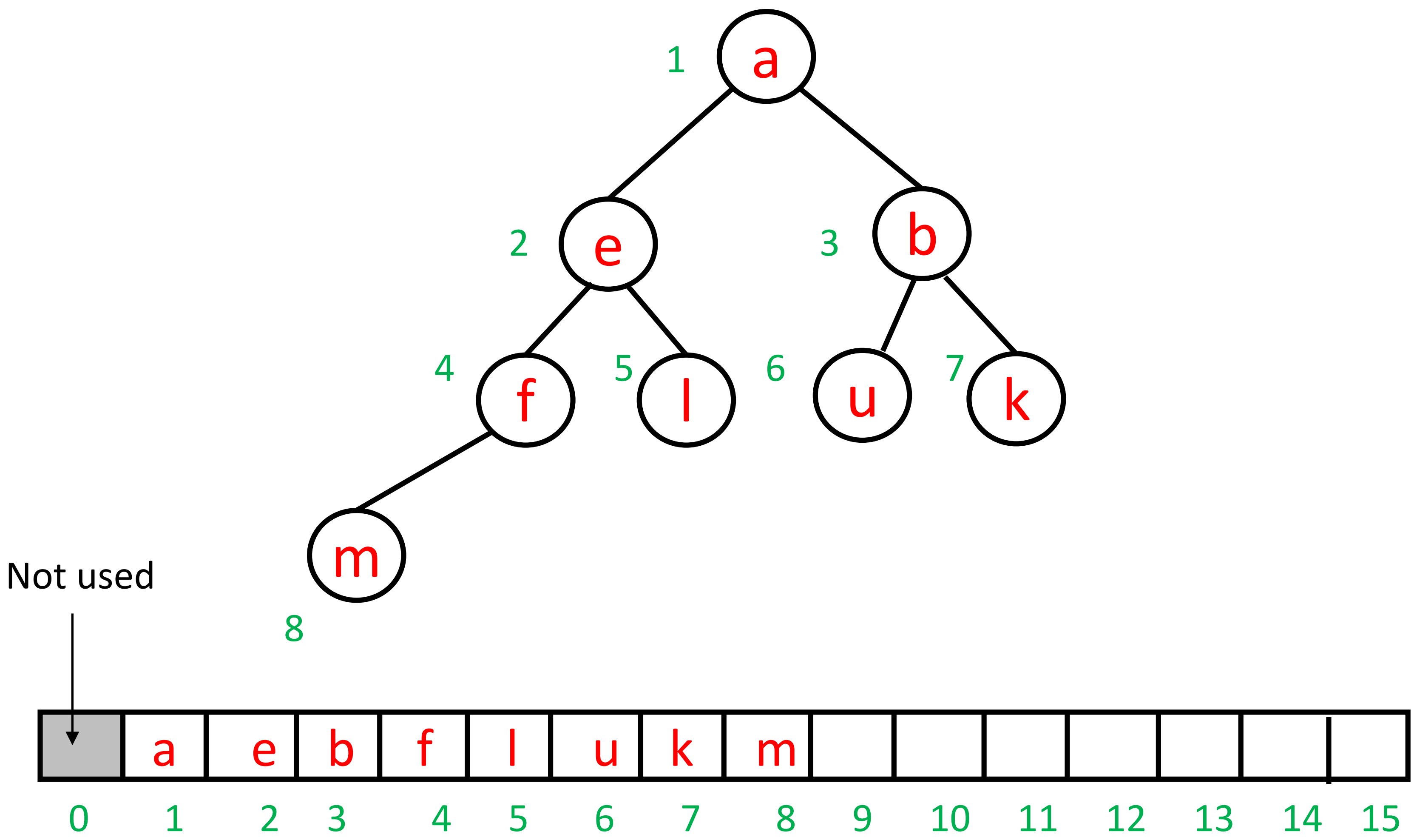
Heap index relations

parent = child / 2

left = 2*parent

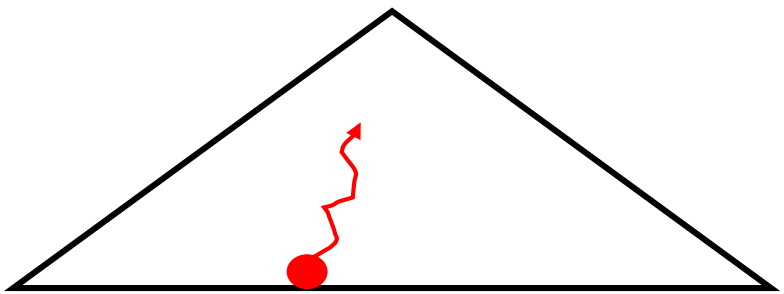
right = 2*parent + 1





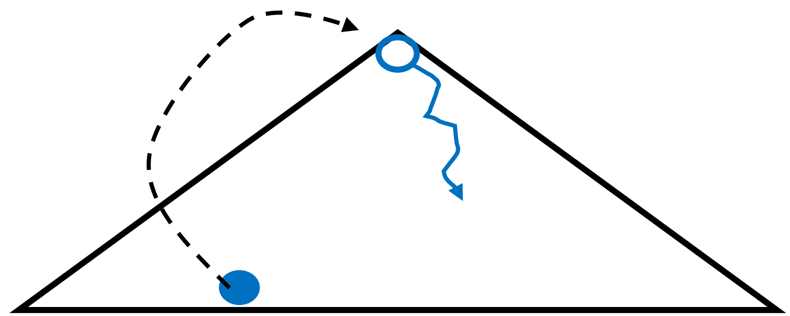
1
a
2
e
3
b
4
f
5
l
6
u
7
k
m
8
Not used
a
e
b
f
l
u
k
m
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

add(element)



"upHeap"

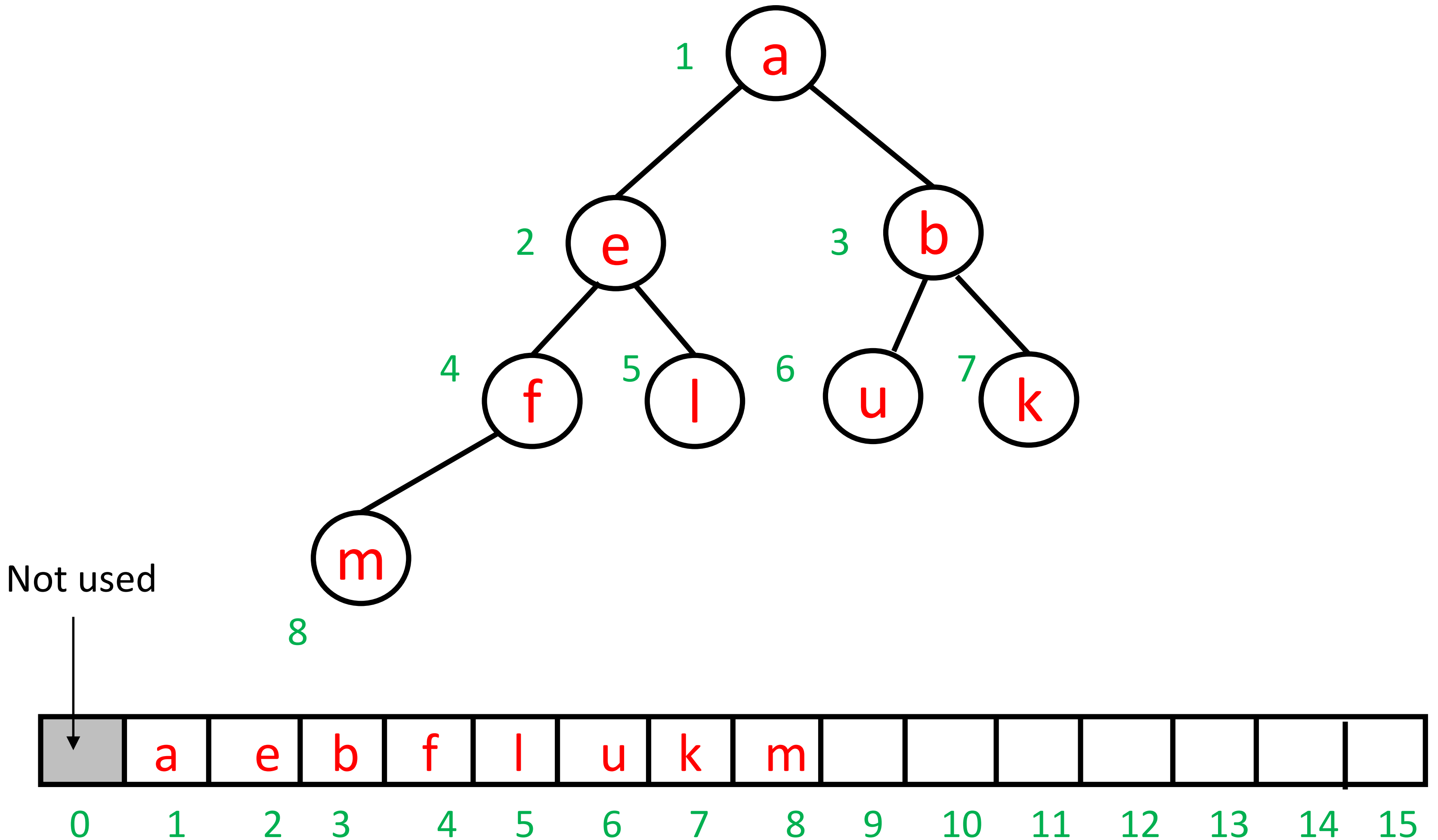
removeMin()



"downHeap"

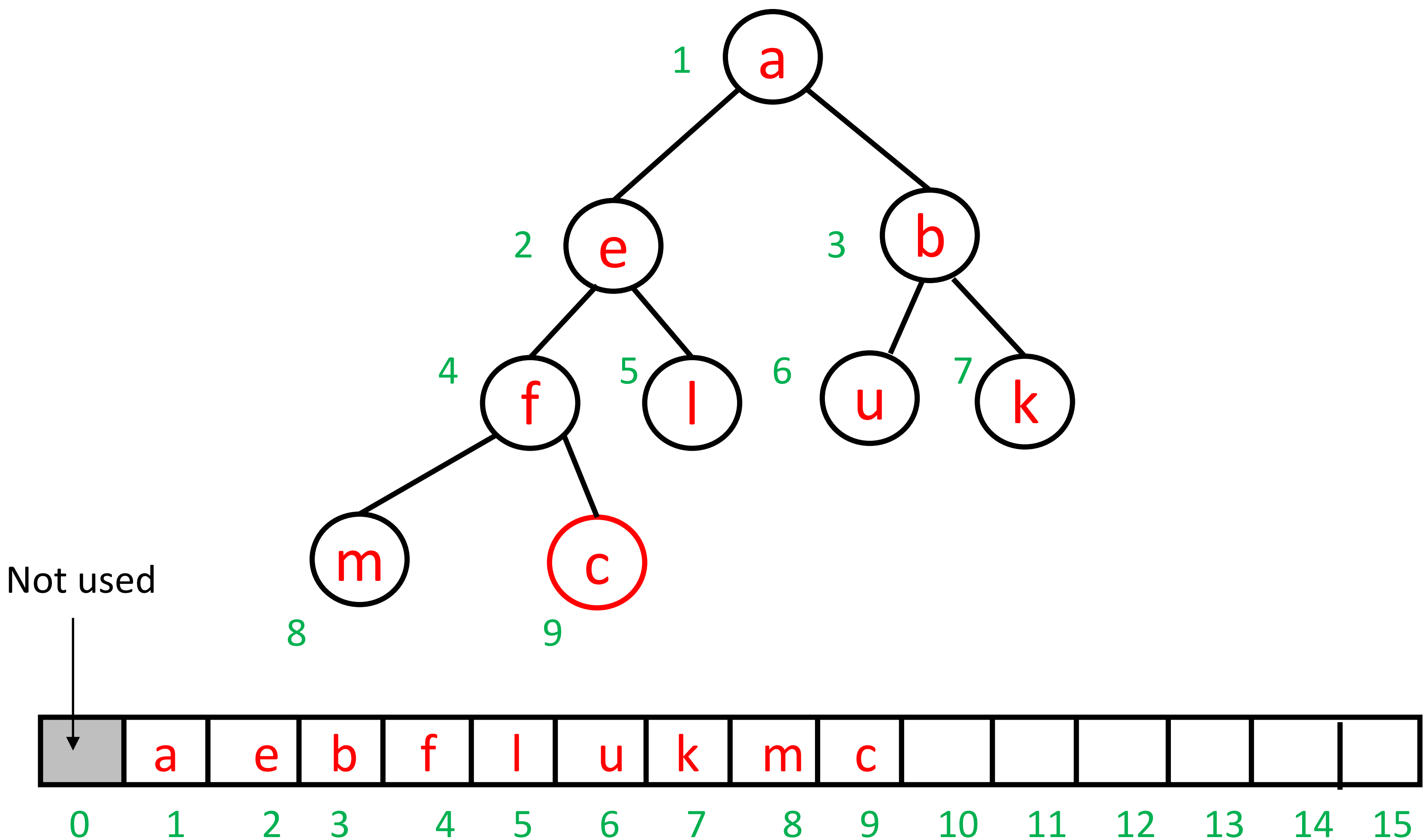
```
add(element ){
    size = size + 1          // number of elements in heap
    heap[ size ] = element   // assuming array
                             //  has room for another element
    i = size

   //  the following is sometimes called "upHeap"

    while ( i > 1  and  heap[i] < heap[ i/2 ]){
        swapElements( i, i/2 )
        i = i/2
    }
}
```

e.g. add(c)



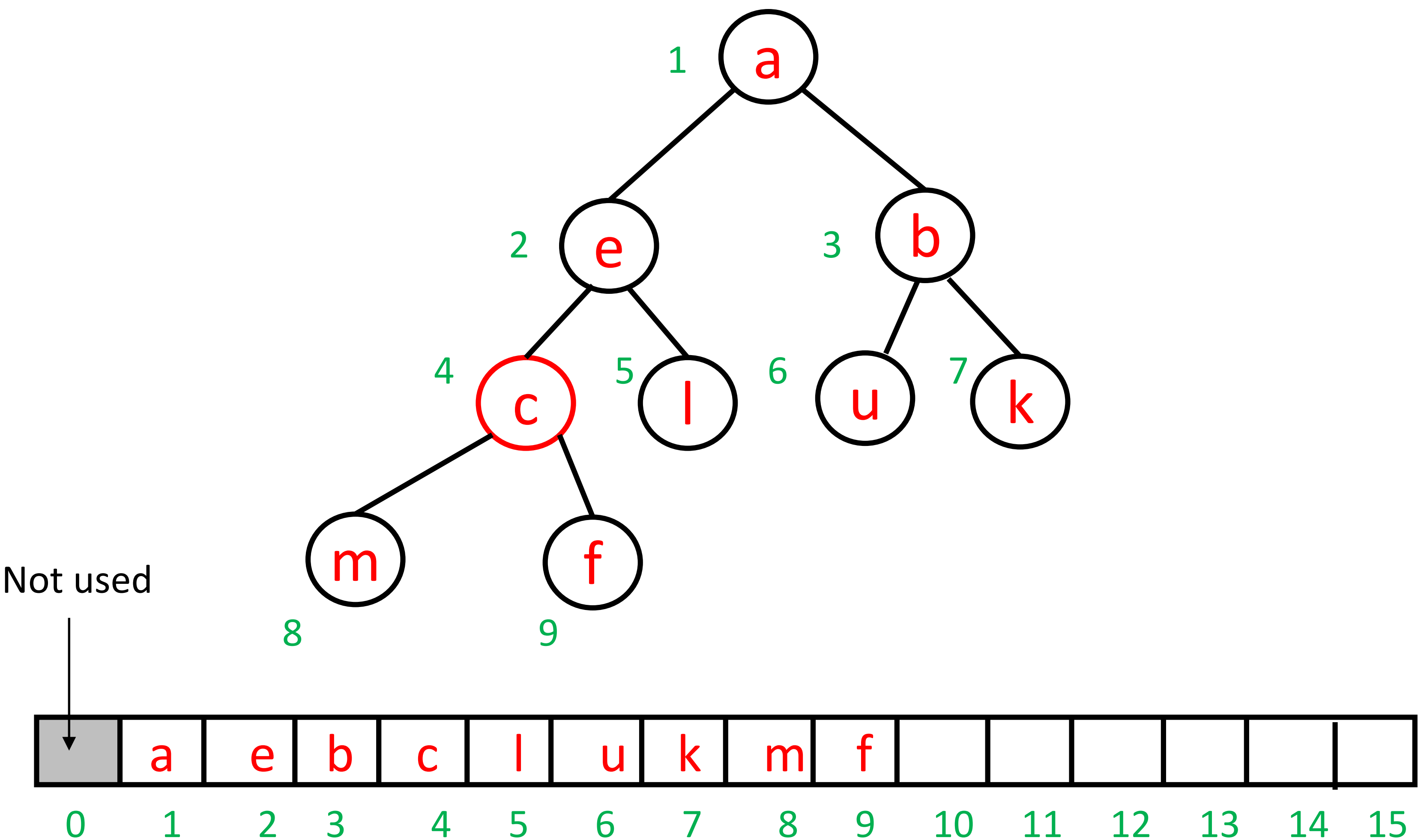
1
a
2
e
3
b
4
f
5
l
6
u
7
k
m
8
Not used
a
e
b
f
l
u
k
m
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

e.g. add(c)



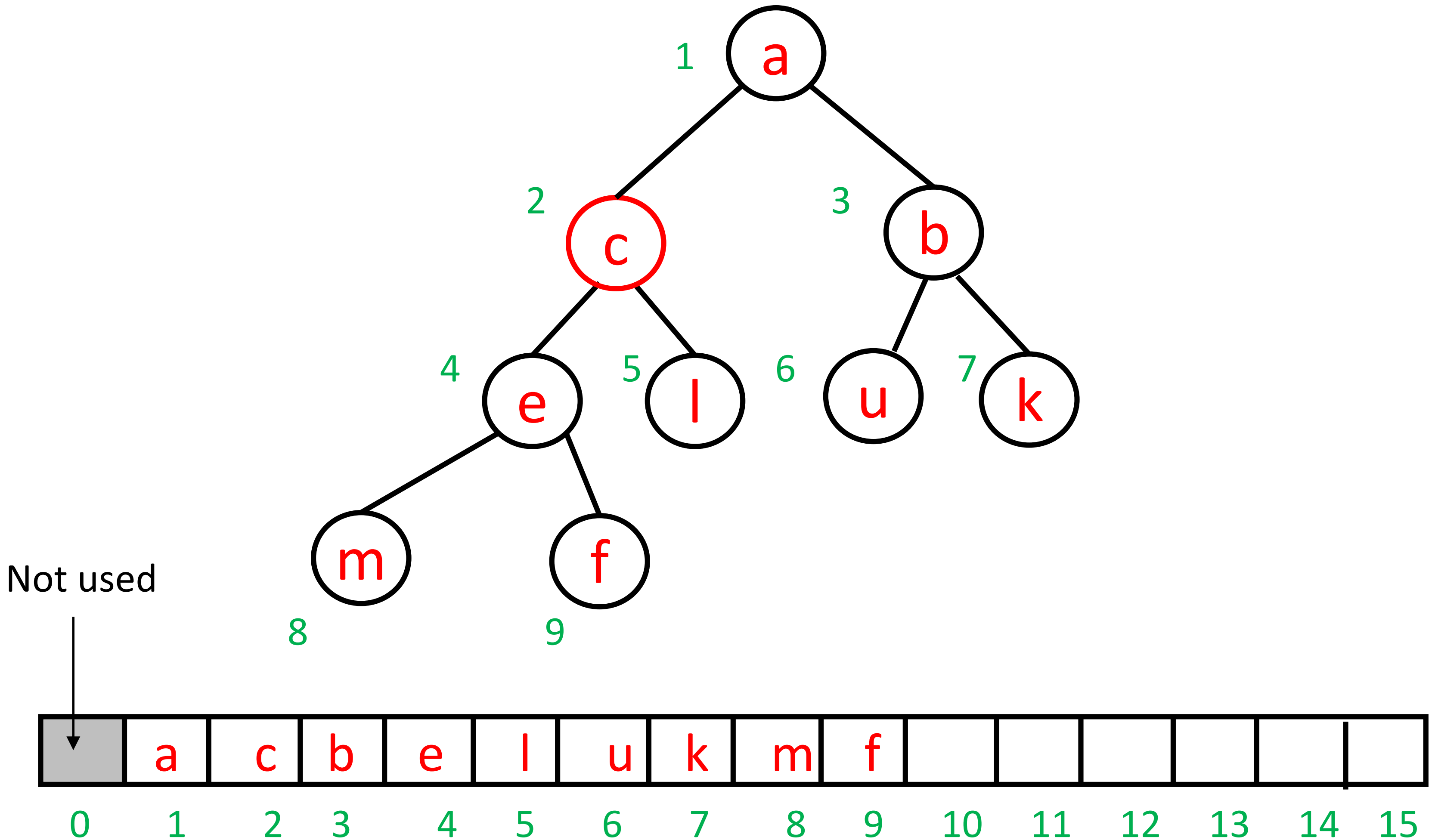
1 a
2 e
3 b
4 f
5 l
6 u
7 k
m
c
8
9
Not used
a e b f l u k m c
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

e.g. add(c)



1
a
2
e
3
b
4
c
5
l
6
u
7
k
m
f
8
9
Not used
a
e
b
c
l
u
k
m
f
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

e.g. add(c)



1 a
2 c
3 b
4 e
5 l
6 u
7 k
m
f
8
9
Not used
a c b e l u k m f
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Given a list with size elements:

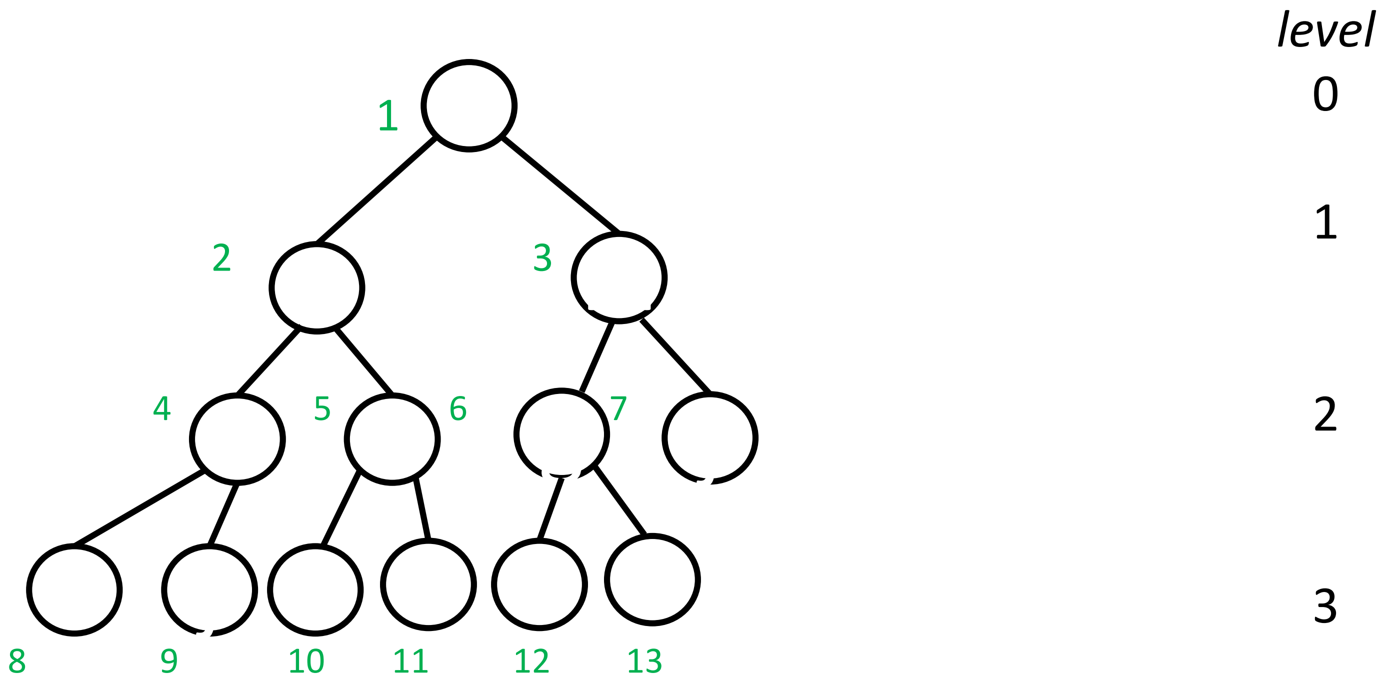
```
buildHeap(list){
    create new heap array              // length >  list.size
    for (k = 0; k < list.size; k++)
        add( list[k] )                 //  add to heap[ ]
}
```

Best case: buildHeap is $\Omega(n)$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	a	e	b	c	l	u	k	m	f						

In the best case, the list is already a heap, and no swaps are necessary.

Worse case of buildHeap?

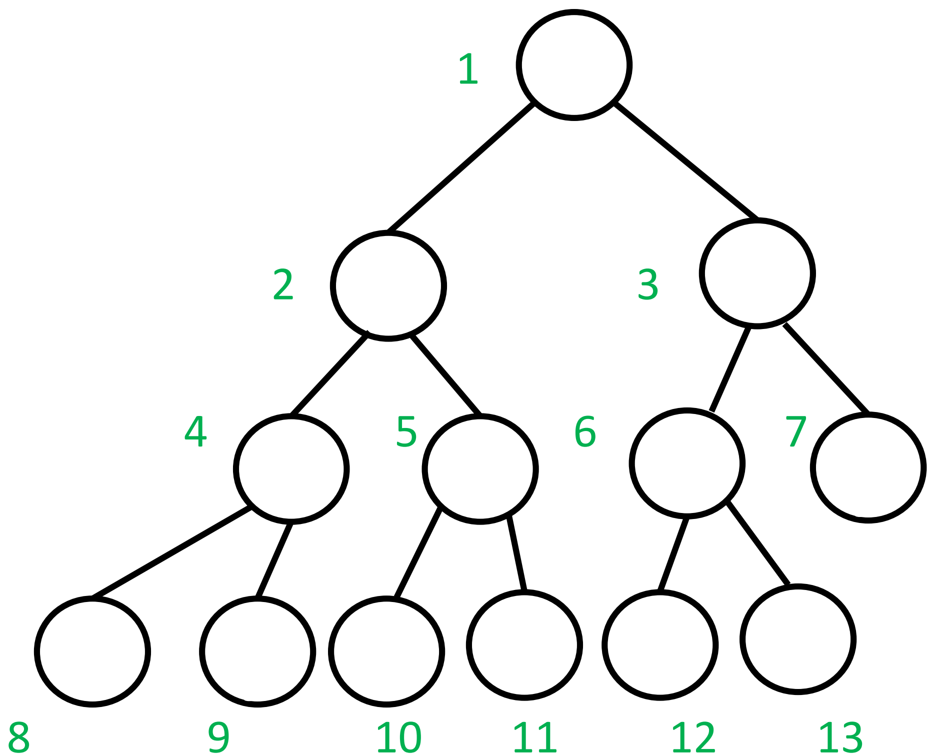


$$2^{level} \leq \ i \ < \ 2^{level+1}$$

$$level \ \leq \quad log_2 i \quad < \ level + 1$$

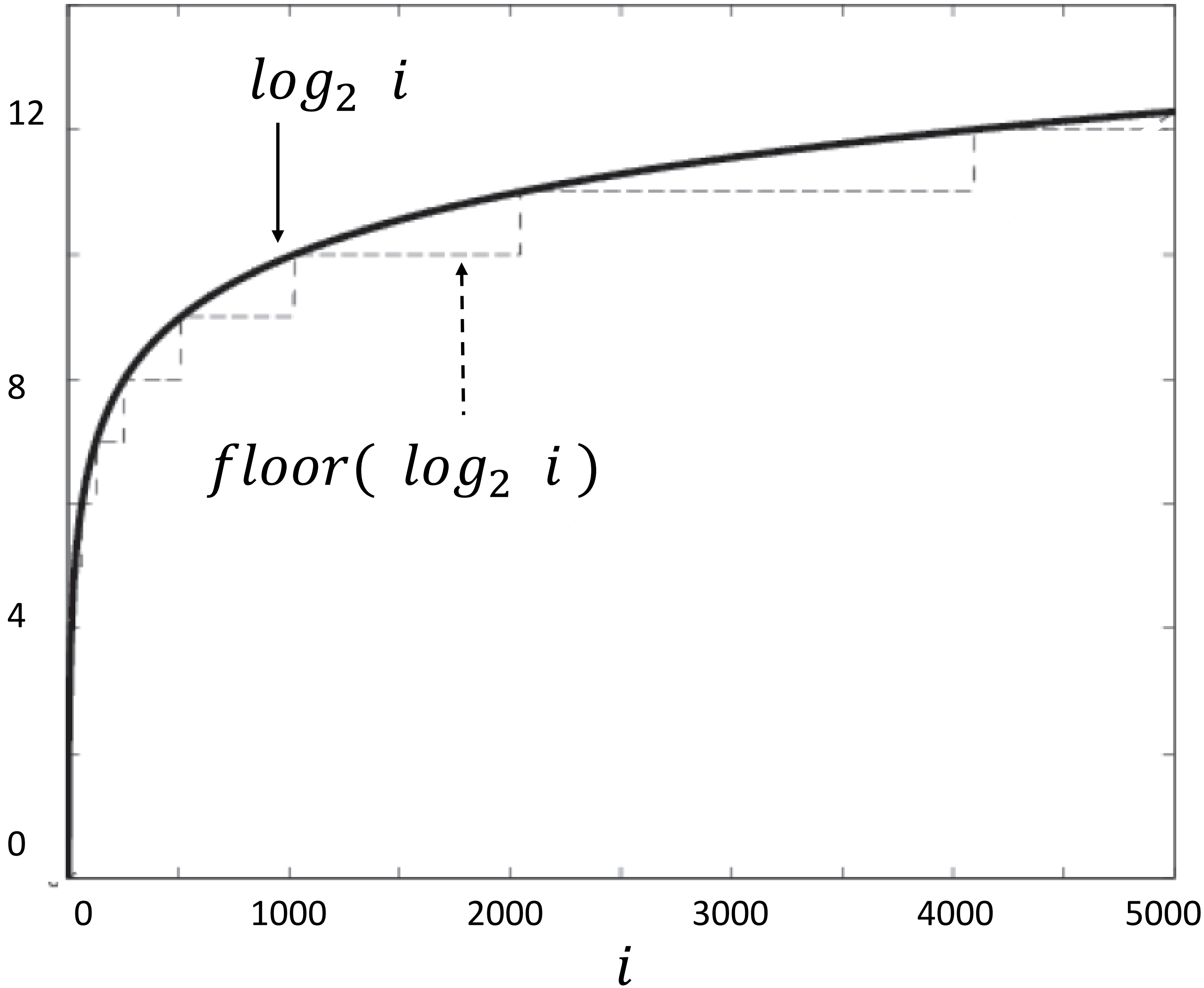
Thus, $\quad level \quad = \quad floor(log_2 i \ )$

Worse case of buildHeap

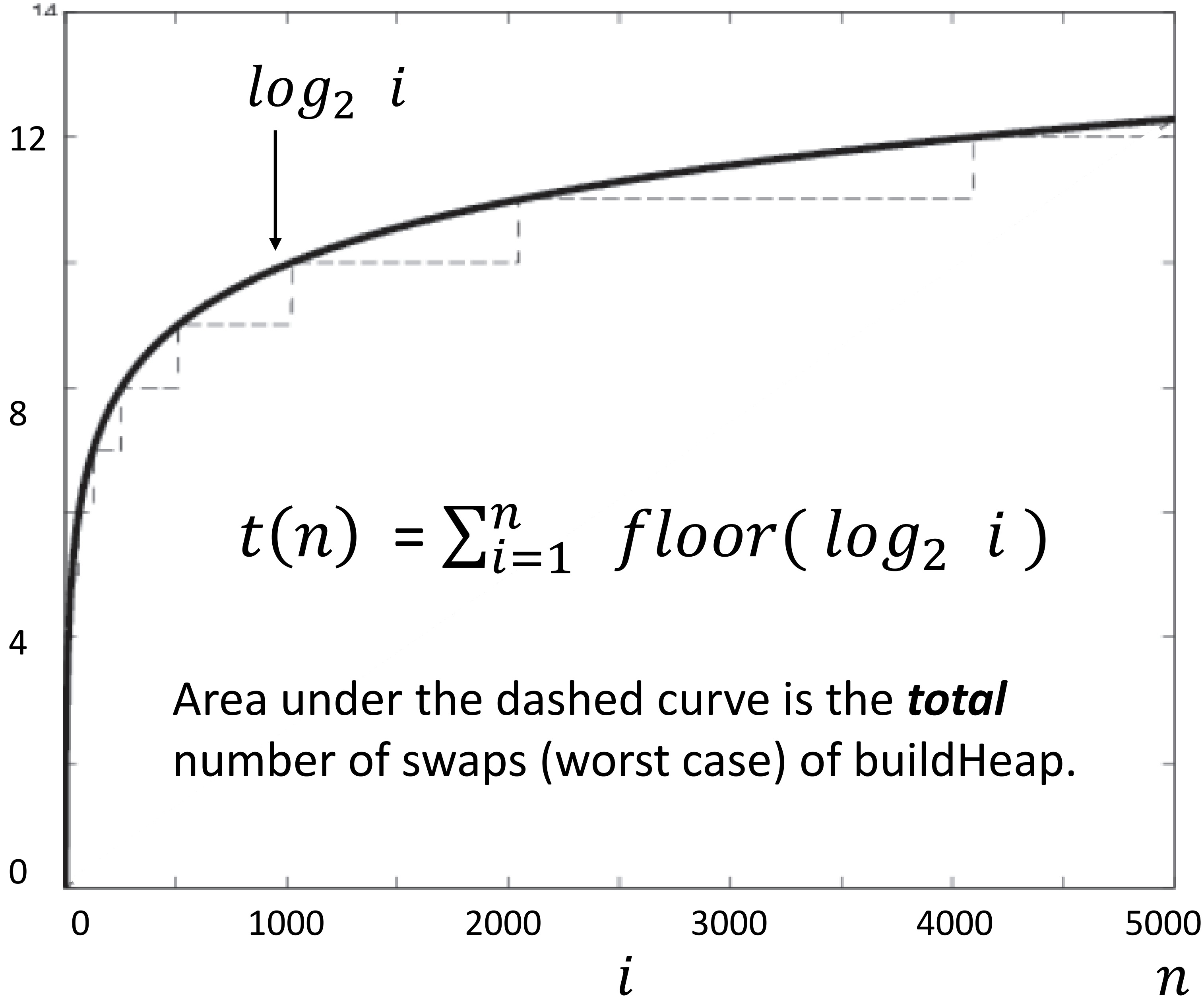


Worst case number of swaps needed to add node i.

$$t(n) = \sum_{i=1}^{n} floor(log_2 \ i)$$

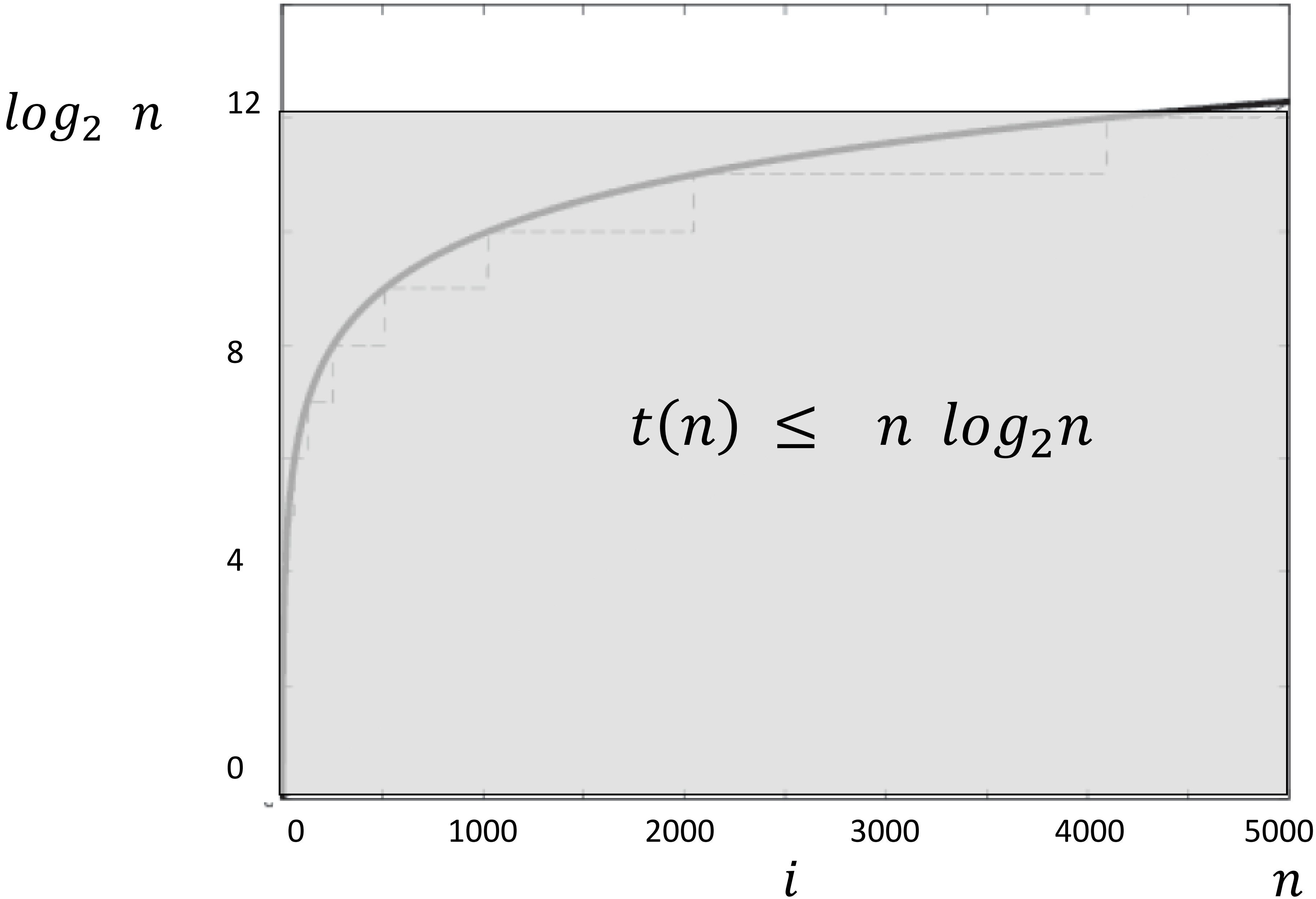


$log_2\ i$
$floor(\ log_2\ i\ )$
12
8
4
0
0
1000
2000
3000
4000
5000
i



$$t(n) = \sum_{i=1}^{n} floor(\, log_2\ i\,)$$

Area under the dashed curve is the ***total*** number of swaps (worst case) of buildHeap.


$log_2\ n$
12
8
4
0
$t(n) \leq n\ log_2 n$
0
1000
2000
3000
4000
5000
i
n

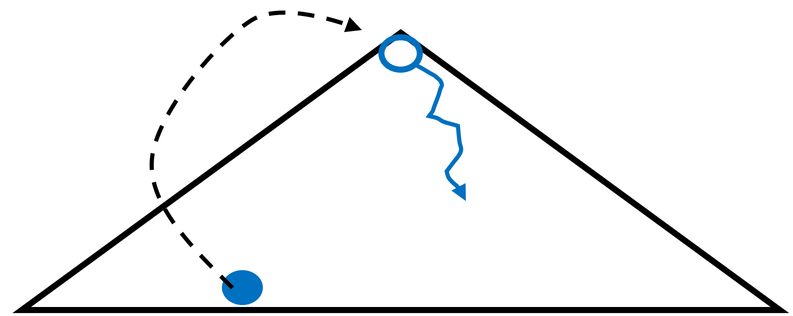
Thus, worst case: buildHeap is $O(n\, log_2\, n)$

Next lecture I will show you a $O(n)$ algorithm.

add(element)

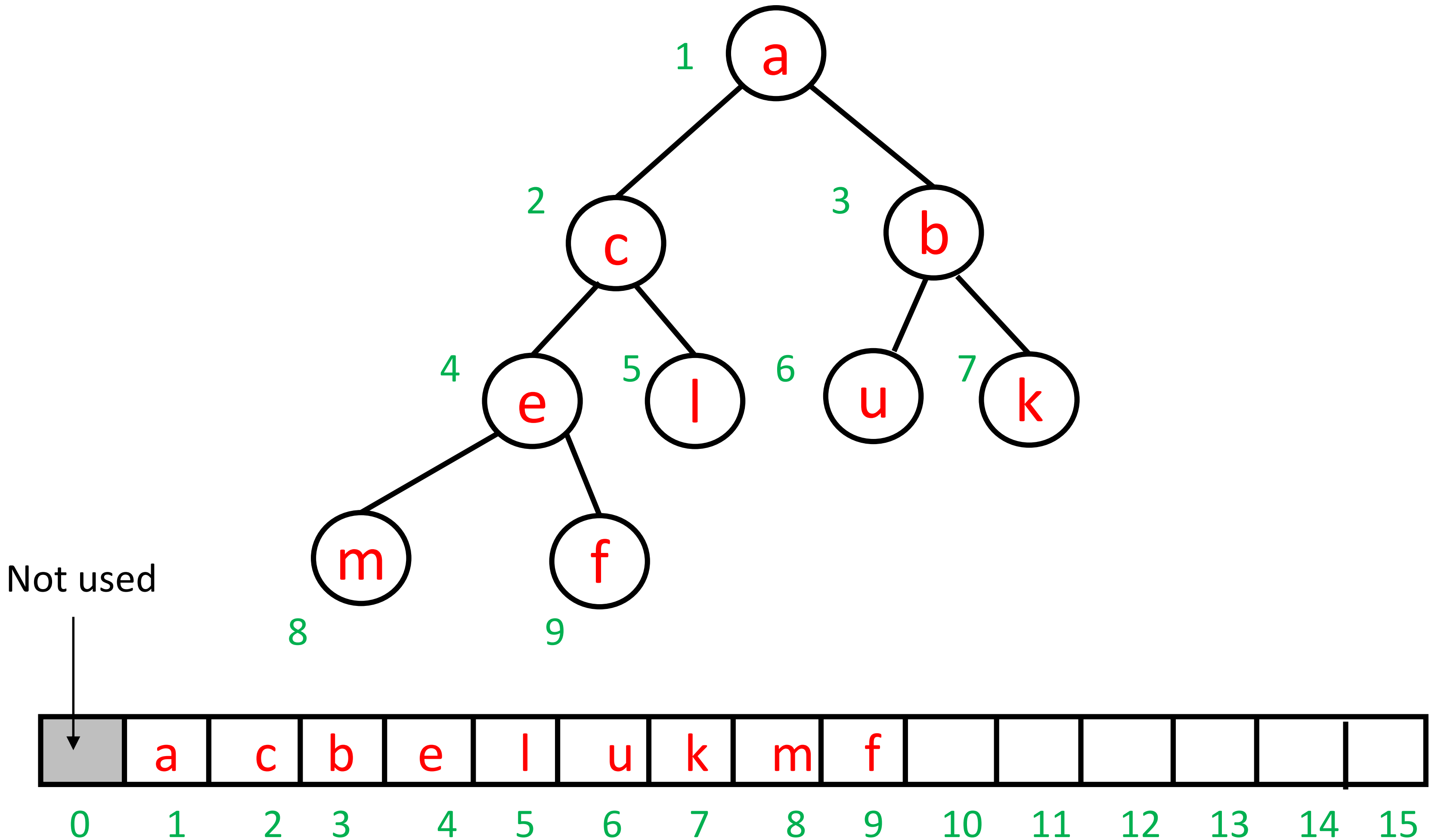
“upHeap”

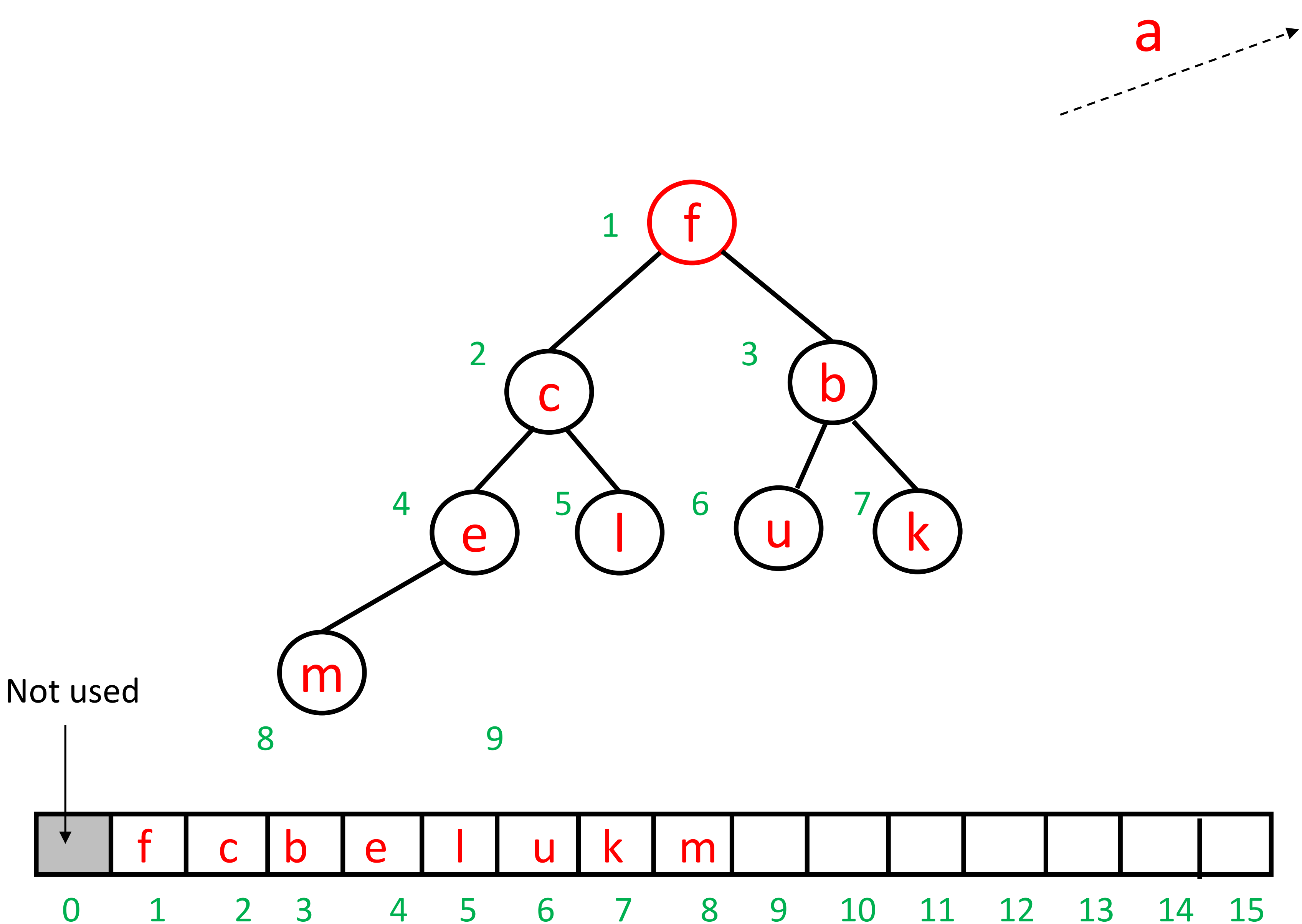
removeMin()

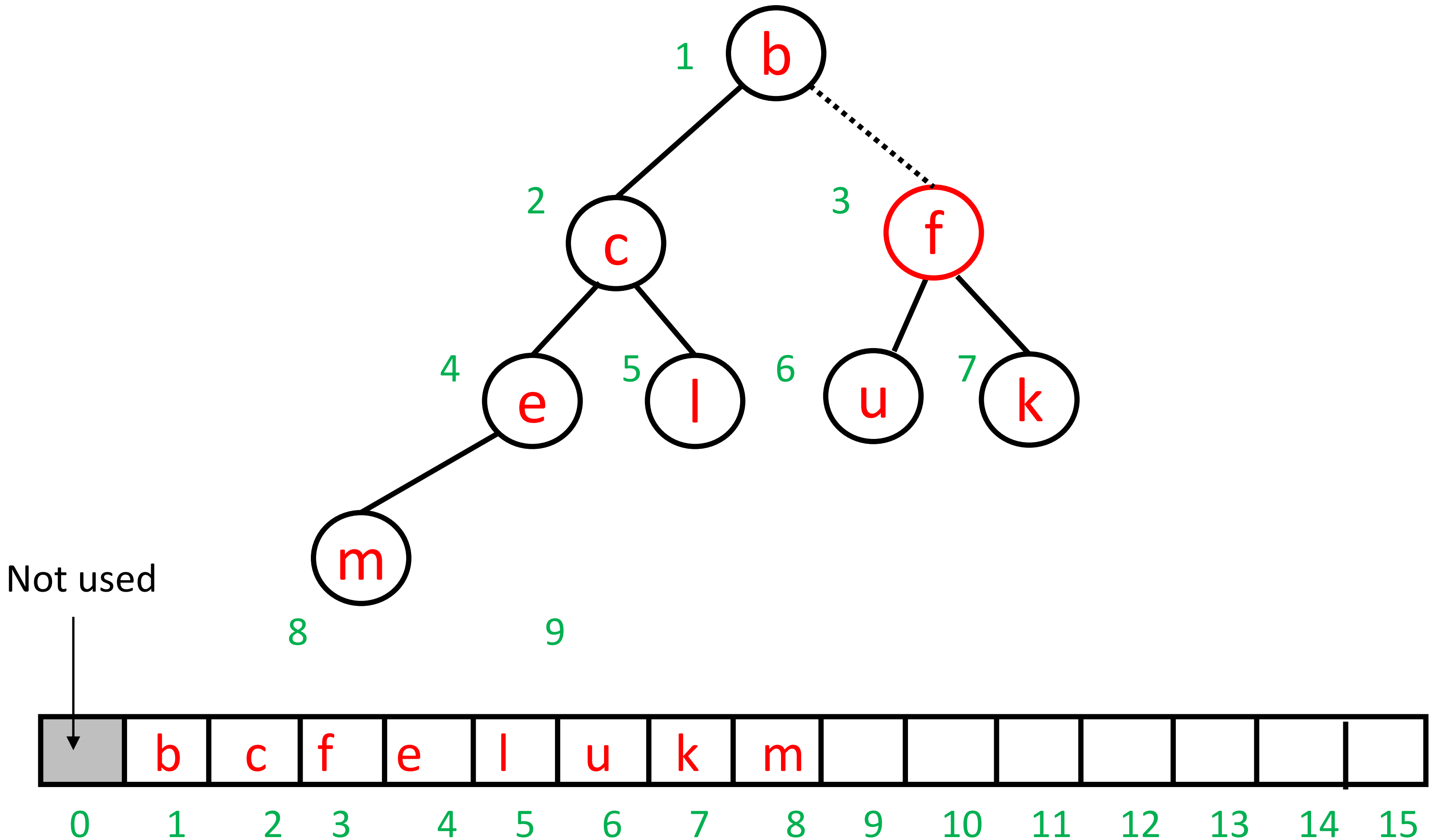


“downHeap”

e.g. removeMin()




a
1
f
2
c
3
b
4
e
5
l
6
u
7
k
m
8
9
Not used
f
c
b
e
l
u
k
m
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



1
b
2
c
3
f
4
e
5
l
6
u
7
k
m
8
9
Not used
b
c
f
e
l
u
k
m
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

removeMin()

Let heap[ ] be the array.

Let size be the number of elements in the heap.

```
removeMin(){
   element = heap[1]          //  heap[0] not used.
   heap[1] = heap[size]
   heap[size] = null
   size = size - 1
   downHeap(1, size)
   return element
}
```

```
downHeap( startIndex , maxIndex ){

  i = startIndex
  while (2*i <= maxIndex){           // if there is a left child
    child = 2*i
    if child < size {                // if there is a right sibling
      if (heap[child + 1] < heap[child])   //  if rightchild < leftchild ?
      child = child + 1
    }
    if (heap[child] < heap[ i ]){    // Do we need to swap with child?
      swapElements(i , child)
      i = child
    }
  }
}
```

Heapsort

Given a list with size elements:

```
heap = buildHeap(list)
for k = 1 to  size{
     list[ size - k ] = heap.removeMin()
}
```

Heapsort

Given a list with size elements:

```
heapsort( list ){
    buildheap(list)
    for i = 1 to size{
        swapElements( heap[1], heap[size + 1 - i])
        downHeap( 1,  size – i )
    }
    return reverse(heap)
}
```

1	2	3	4	5	6	7	8	9
a	d	b	e	l	u	k	f	w |

1	2	3	4	5	6	7	8	9	
a	d	b	e	l	u	k	f	w	|
w	d	b	e	l	u	k	f |	a	

1	2	3	4	5	6	7	8		9	
a	d	b	e	l	u	k	f		w	|
w	d	b	e	l	u	k	f	|	a	
b	d	w	e	l	u	k	f	|	a	

```
1    2    3    4    5    6    7    8    9
------------------------------------------------
a    d    b    e    l    u    k    f    w |
w    d    b    e    l    u    k    f  | a
b    d    w    e    l    u    k    f  | a
b    d    k    e    l    u    w    f  | a
```

1	2	3	4	5	6	7	8	9	
a	d	b	e	l	u	k	f	w	|
b	d	k	e	l	u	w	f |	a	

```
1   2   3   4   5   6   7   8   9
---------------------------------------
a   d   b   e   l   u   k   f   w |
b   d   k   e   l   u   w   f | a
f   d   k   e   l   u   w | b   a
```

```
1   2   3   4   5   6   7   8   9
-------------------------------------
a   d   b   e   l   u   k   f   w |
b   d   k   e   l   u   w   f | a
f   d   k   e   l   u   w | b   a
d   f   k   e   l   u   w | b   a
```

```
1   2   3   4   5   6   7   8   9
-----------------------------------
a   d   b   e   l   u   k   f   w |
b   d   k   e   l   u   w   f | a
f   d   k   e   l   u   w | b   a
d   f   k   e   l   u   w | b   a
d   e   k   f   l   u   w | b   a
```

1	2	3	4	5	6	7	8	9
a	d	b	e	l	u	k	f	w |
b	d	k	e	l	u	w	f |	a
d	e	k	f	l	u	w |	b	a

```
1   2   3   4   5   6   7   8   9
------------------------------------
a   d   b   e   l   u   k   f   w |
b   d   k   e   l   u   w   f | a
d   e   k   f   l   u   w | b   a
e   f   k   w   l   u | d   b   a
```

```
1   2   3   4   5   6   7   8   9
-------------------------------------
a   d   b   e   l   u   k   f   w |
b   d   k   e   l   u   w   f | a
d   e   k   f   l   u   w | b   a
e   f   k   w   l   u | d   b   a
f   l   k   w   u | e   d   b   a
k   l   u   w | f   e   d   b   a
l   w   u | k   f   e   d   b   a
u   w | l   k   f   e   d   b   a
w | u   l   k   f   e   d   b   a
w   u   l   k   f   e   d   b   a
```

Heapsort

```
heapsort(list){
    buildheap(list)
    for i = 1 to size{
        swapElements( heap[1], heap[size + 1 - i])
        downHeap( 1,  size - i)
    }
    return reverse(heap)
}
```

Best and worst case of heapsort ?

See Exercises.