F10 - rekursion och sortering

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin
niclas.borlin@cs.umu.se

Datavetenskap, Umeå universitet

2023-10-13 Fre

Rekursion

Rekursion

- Det finns inget som hindrar att en funktion anropar sig själv
  - Detta kallas rekursion
- För att rekursionen skall terminera (avslutas), måste det
  1. finnas ett eller flera stoppvillkor (basfall) och
  2. varje rekursivt anrop måste ta oss minst ett steg närmare ett stoppvillkor

Ett exempel (2)

- En multiplikation går att se som en sekvens av additioner
  - $m \cdot n = \underbrace{m + m + \cdots + m}_{n}$
- En rekursiv algoritm `mult(m, n)` skulle kunna se ut så här:

```
1. Om n = 1
    1.1 Returnera m
2. annars
    2.1 Returnera mult(m, n - 1) + m
```

Ett exempel (2)

- En multiplikation går att se som en sekvens av additioner
  - $m \cdot n = \underbrace{m + m + \cdots + m}_{n}$
- En rekursiv algoritm `mult(m, n)` skulle kunna se ut så här:

```
1. Om n = 1
    1.1 Returnera m
2. annars
    2.1 Returnera mult(m, n - 1) + m
```

- Basfallet är `n = 1`

Ett exempel (2)

- En multiplikation går att se som en sekvens av additioner
  - $m \cdot n = \underbrace{m + m + \cdots + m}_{n}$
- En rekursiv algoritm `mult(m, n)` skulle kunna se ut så här:

```
1. Om n = 1
    1.1 Returnera m
2. annars
    2.1 Returnera mult(m, n - 1) + m
```

- Basfallet är `n = 1`
- I det rekursiva fallet anropar vi `mult` med värdena `m` och `n - 1` (och adderar sedan `m`)
  - Vi kommer ett steg närmare basfallet

mult, körning

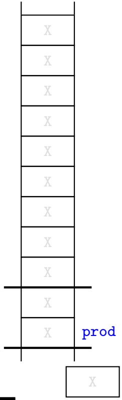
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```

X
X
X
X
X
X
X
X
X
X
X
X

[redacted]

mult, körning

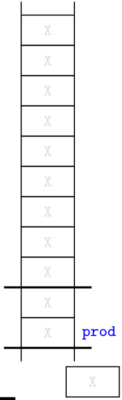
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

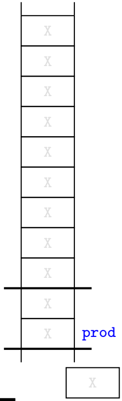
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

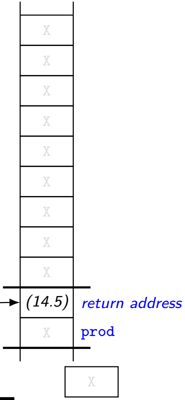
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

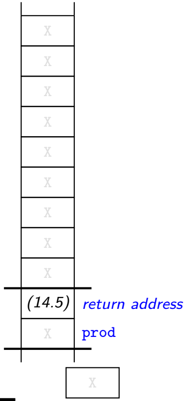
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

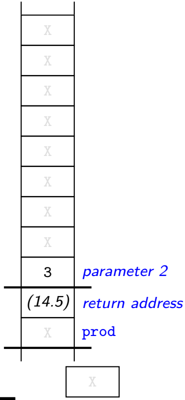
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

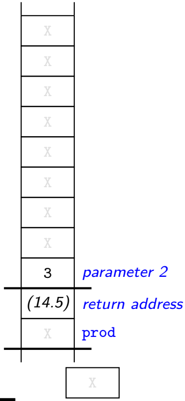
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

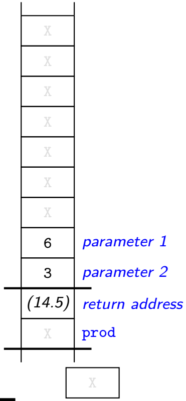
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

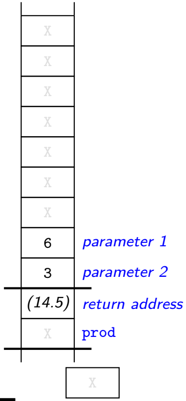
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

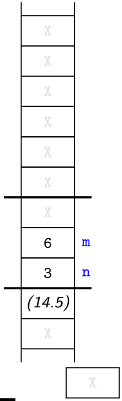
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

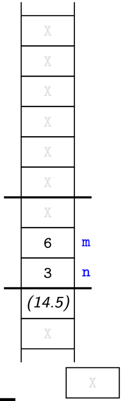
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

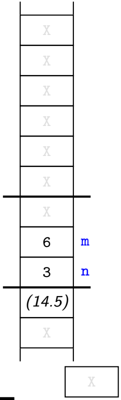
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```

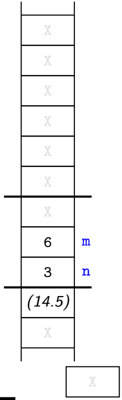


[redacted]

14

mult, körning

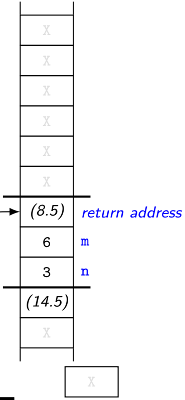
```
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

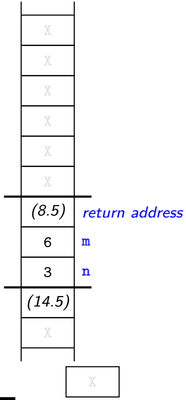
mult, körning

```
1   #include <stdio.h>
2
3   int mult(int m, int n)
4   {
5       if (n == 1) {
6           return m;
7       } else {
8           return mult(m, n - 1) + m;
9       }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



X

X

X

X

X

X

(8.5) return address

6 m

3 n

(14.5)

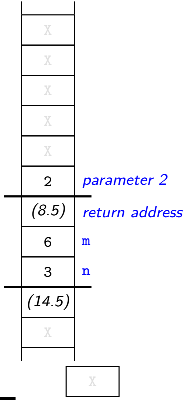
X

X

[redacted]

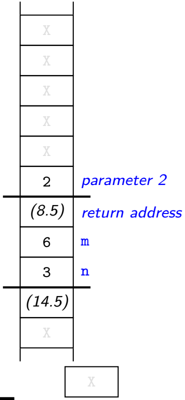
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

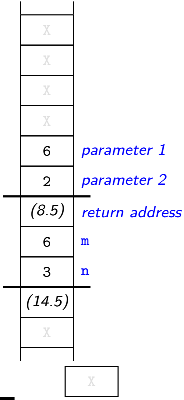
```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



[redacted]

mult, körning

```
1  #include <stdio.h>
2
3  int mult(int m, int n)
4  {
5      if (n == 1) {
6          return m;
7      } else {
8          return mult(m, n - 1) + m;
9      }
10 }
11
12 int main(void)
13 {
14     int prod = mult(6, 3);
15     printf("6 x 3 = %d\n", prod);
16     return 0;
17 }
```

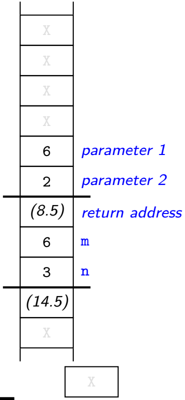


X
X
X
X
6 parameter 1
2 parameter 2
(8.5) return address
6 m
3 n
(14.5)
X
X

[redacted]

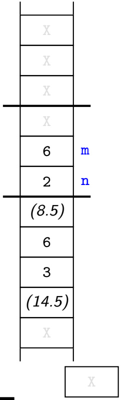
mult, körning

```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



mult, körning

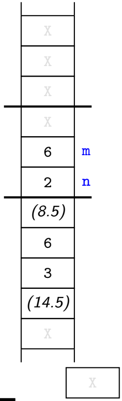
```
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

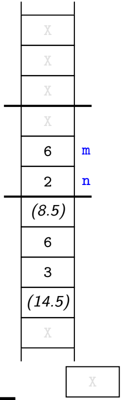
```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



[redacted]

mult, körning

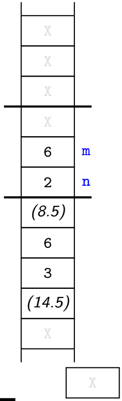
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

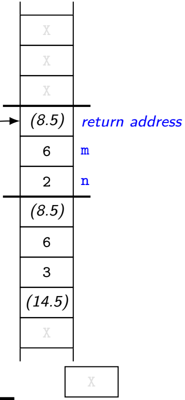
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

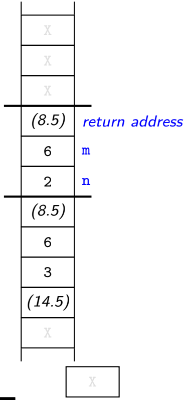
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

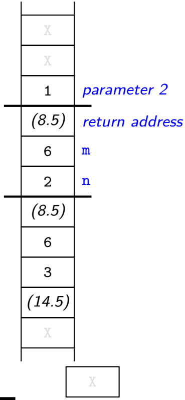
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

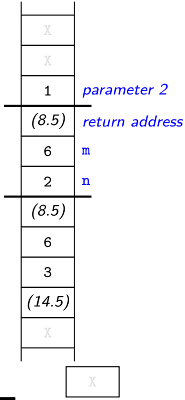
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



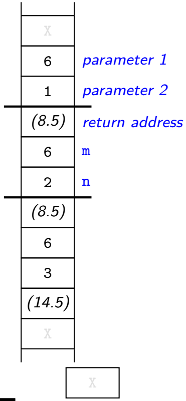
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



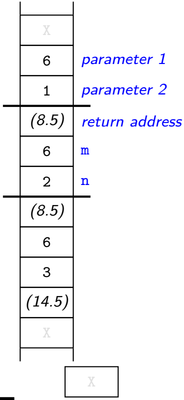
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



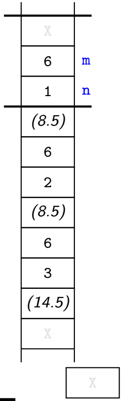
X	
6	parameter 1
1	parameter 2
(8.5)	return address
6	m
2	n
(8.5)	
6	
3	
(14.5)	
X	

X

[redacted]

mult, körning

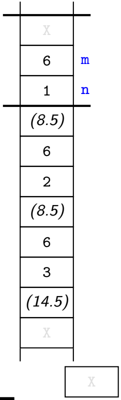
```
1   #include <stdio.h>
2
3   int mult(int m, int n)
4   {
5       if (n == 1) {
6           return m;
7       } else {
8           return mult(m, n - 1) + m;
9       }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



[redacted]

mult, körning

```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



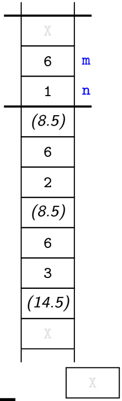
X	
6	m
1	n
(8.5)	
6	
2	
(8.5)	
6	
3	
(14.5)	
X	

X

[redacted]

mult, körning

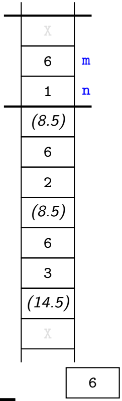
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

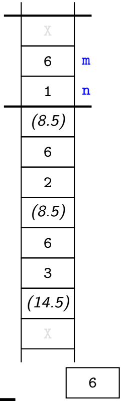
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



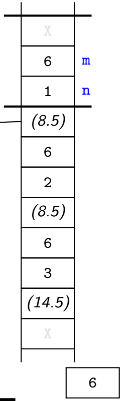
X
6 m
1 n
(8.5)
6
2
(8.5)
6
3
(14.5)
X

6

[redacted]

mult, körning

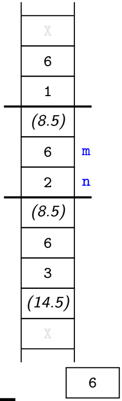
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

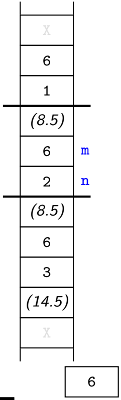
mult, körning

```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



mult, körning

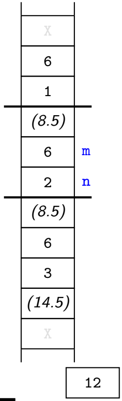
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

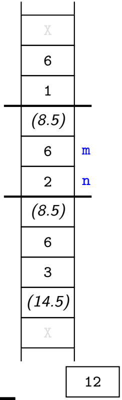
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

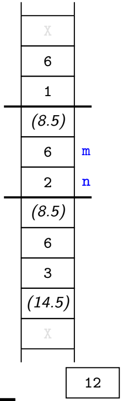
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

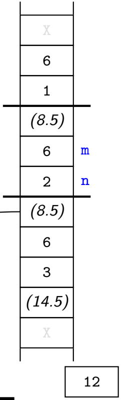
```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



[redacted]

mult, körning

```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```



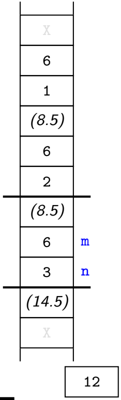
X	
6	
1	
(8.5)	
6	m
2	n
(8.5)	
6	
3	
(14.5)	
X	

12

[redacted]

mult, körning

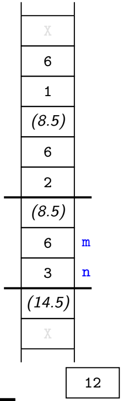
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

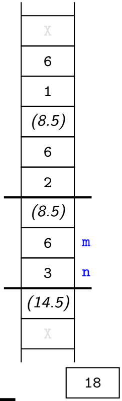
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

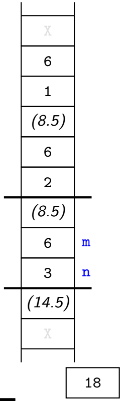
mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



mult, körning

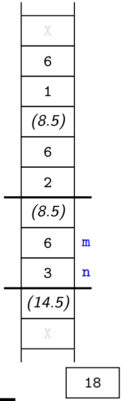
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

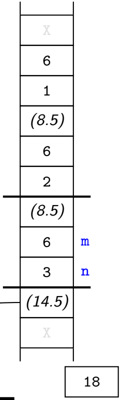
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 * 3 = %d\n", prod);
    return 0;
}
```



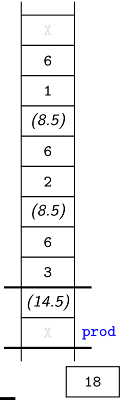
X	
6	
1	
(8.5)	
6	
2	
(8.5)	
6	m
3	n
(14.5)	
X	

18

[redacted]

mult, körning

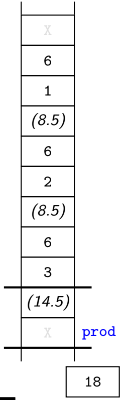
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

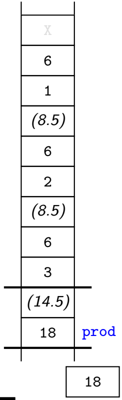
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

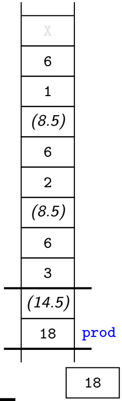
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

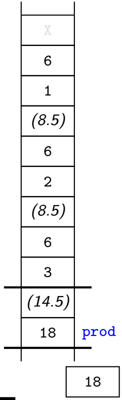
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



[redacted]

mult, körning

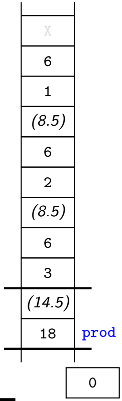
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```



```
6 x 3 = 18
```

mult, körning

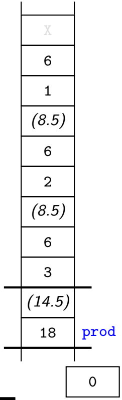
```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```

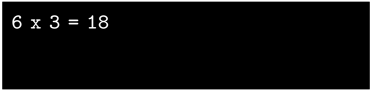


```
6 x 3 = 18
```

mult, körning

```c
#include <stdio.h>

int mult(int m, int n)
{
    if (n == 1) {
        return m;
    } else {
        return mult(m, n - 1) + m;
    }
}

int main(void)
{
    int prod = mult(6, 3);
    printf("6 x 3 = %d\n", prod);
    return 0;
}
```





```
6 x 3 = 18
```

mult, körning

```
 1  #include <stdio.h>
 2
 3  int mult(int m, int n)
 4  {
 5      if (n == 1) {
 6          return m;
 7      } else {
 8          return mult(m, n - 1) + m;
 9      }
10  }
11
12  int main(void)
13  {
14      int prod = mult(6, 3);
15      printf("6 x 3 = %d\n", prod);
16      return 0;
17  }
```

X

6

1

(8.5)

6

2

(8.5)

6

3

(14.5)

18

0

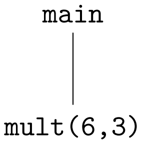
```
6 x 3 = 18
```

1

Anropsträd

`main`

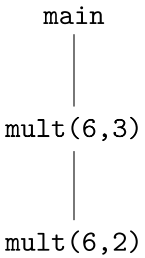
Anropsträd



main

mult(6,3)

Anropsträd

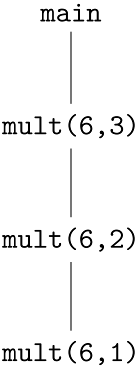


main

mult(6,3)

mult(6,2)

Anropsträd

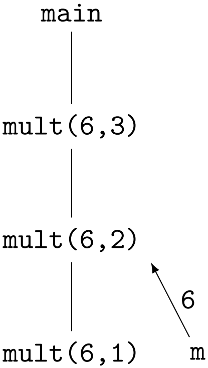

main

mult(6,3)

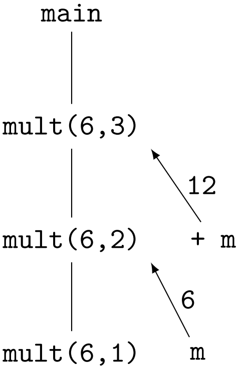
mult(6,2)

mult(6,1)

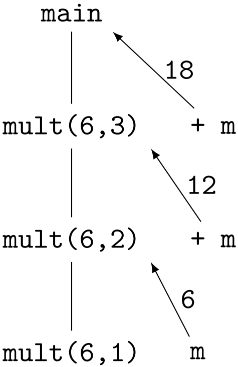
Anropsträd



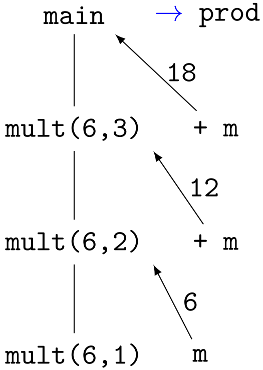
Anropsträd



Anropsträd



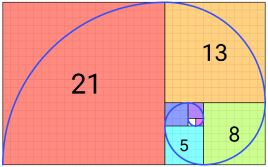
Anropsträd

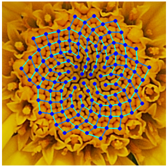


Ett exempel till

- Fibonacci-sekvensen är definierad som en rekursiv sekvens:
  - $F_0 = 0$
  - $F_1 = 1$
  - $F_n = F_{n-1} + F_{n-2}$
- Sekvensen blir
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

och den dyker upp på många ställen i naturen[1]





[1] https://en.wikipedia.org/wiki/Fibonacci_number

Fibonacci-sekevensen

- Följande kod beräknar det n:te talet i Fibonacci-sekvensen:

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

Fibonacci-sekevensen

- Följande kod beräknar det n:te talet i Fibonacci-sekvensen:

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

- Basfallen är `n = 0` och `n = 1`

Fibonacci-sekevensen

- Följande kod beräknar det n:te talet i Fibonacci-sekvensen:

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

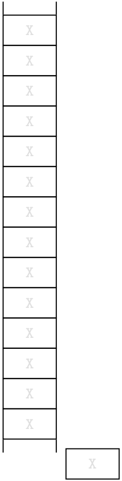
- Basfallen är `n = 0` och `n = 1`
- I de rekursiva fallen anropar vi `fib` med värdena `n-1` och `n-2`
  - Vi kommer minst ett steg närmare basfallen

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

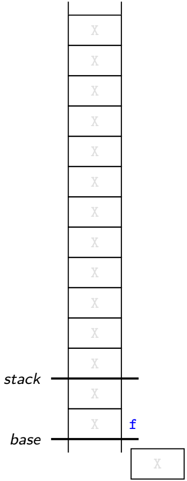


fib, körning

code/fib.c

```
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

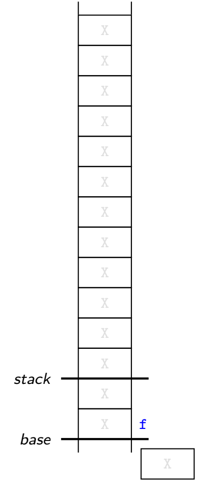


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

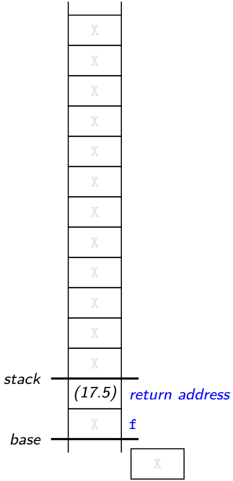


fib, körning

code/fib.c

```
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

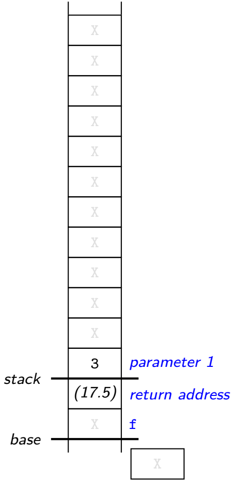


fib, körning

code/fib.c

```
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]



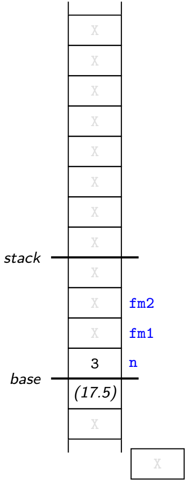
X
X
X
X
X
X
X
X
X
X
X
3 *parameter 1*
stack
(17.5) *return address*
X f
base
X

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

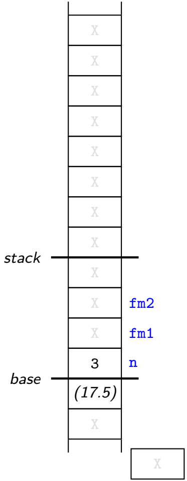


fib, körning

code/fib.c

```
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

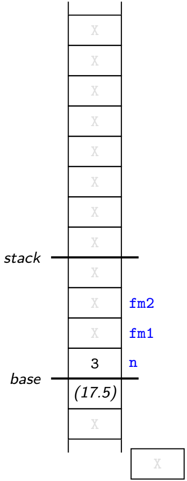


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

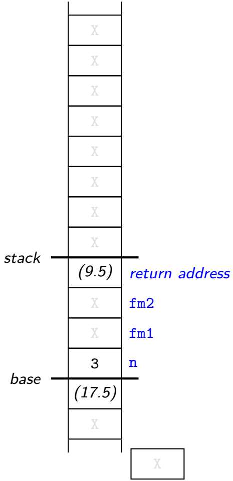
[redacted]



fib, körning

code/fib.c

```
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```



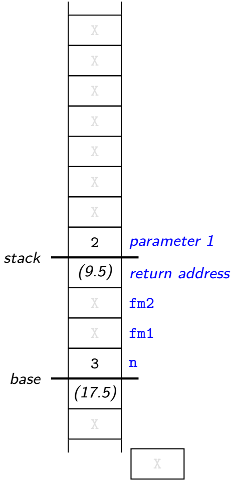
[redacted]

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

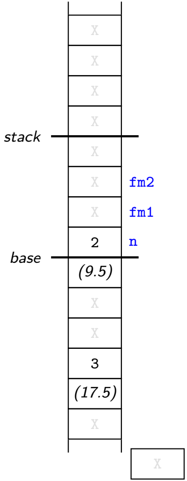
[redacted]



fib, körning

code/fib.c

```
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

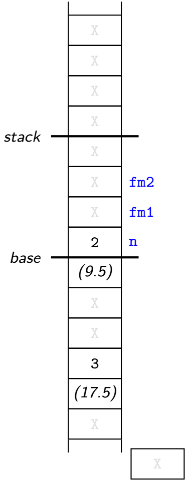


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

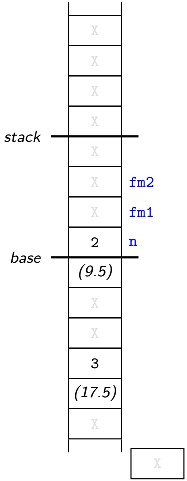
[redacted]



fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



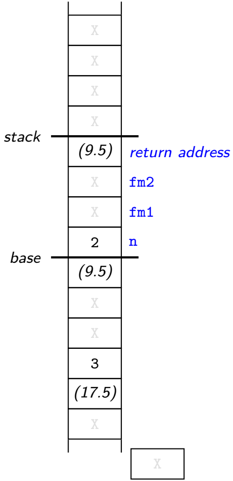
[redacted]

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

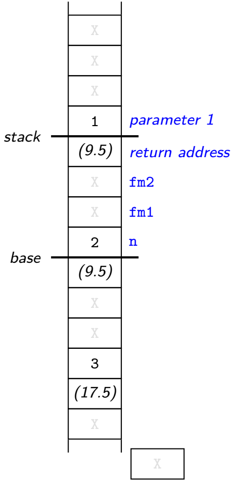


fib, körning

code/fib.c

```
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

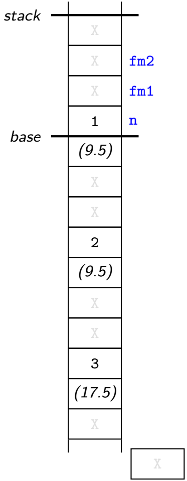


fib, körning

code/fib.c

```
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

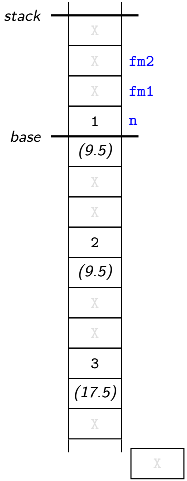


fib, körning

code/fib.c

```
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]



stack

X

X fm2

X fm1

1 n

base

(9.5)

X

X

2

(9.5)

X

X

3

(17.5)

X

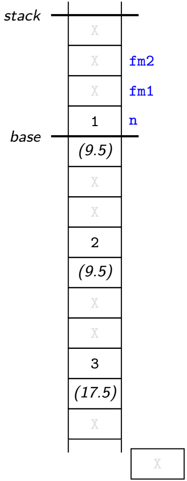
X

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

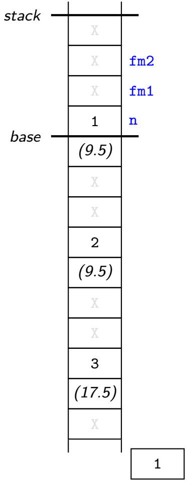


fib, körning

code/fib.c

```c
1  #include <stdio.h>
2
3  int fib(int n)
4  {
5      int fm1, fm2;
6      if (n < 2) {
7          return n;
8      } else {
9          fm1 = fib(n - 1);
10         fm2 = fib(n - 2);
11         return fm1 + fm2;
12     }
13 }
14
15 int main(void)
16 {
17     int f = fib(3);
18     printf("fib(3) = %d\n", f);
19     return 0;
20 }
```

[redacted]

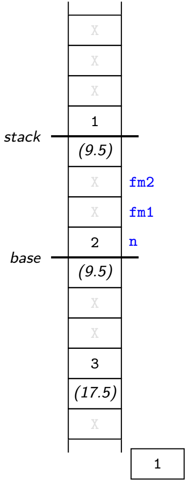


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

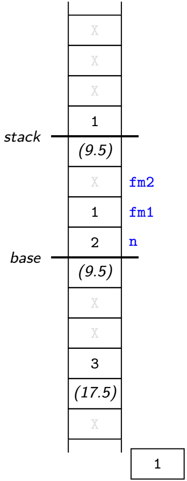
[redacted]



fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



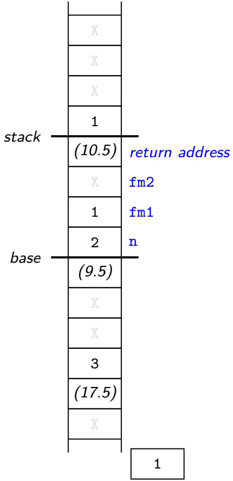
[redacted]

fib, körning

code/fib.c

```c
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

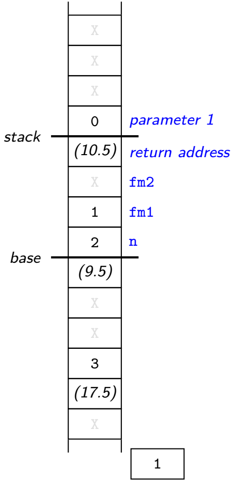
[redacted]



fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



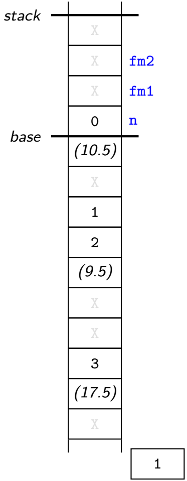
[redacted]

fib, körning

code/fib.c

```
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

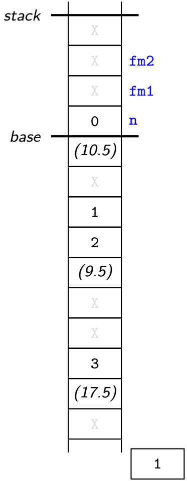
[redacted]



fib, körning

code/fib.c

```c
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

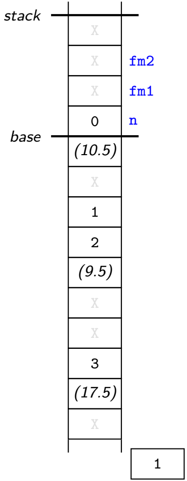


[redacted]

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

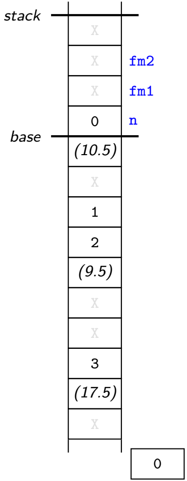


[redacted]

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



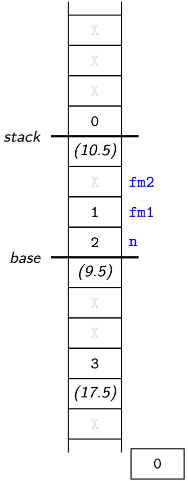
[redacted]

fib, körning

code/fib.c

```c
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

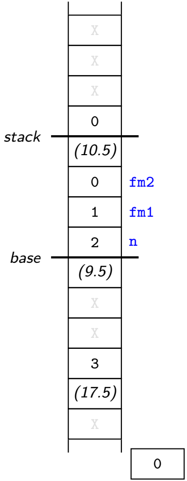


fib, körning

code/fib.c

```
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

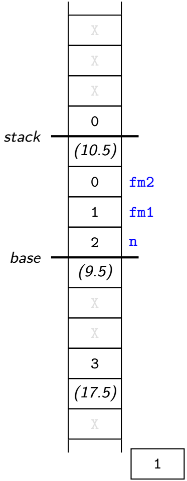


fib, körning

code/fib.c

```
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

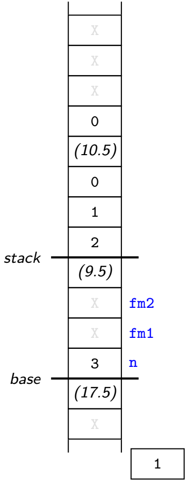


`fib`, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

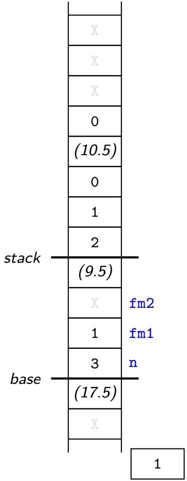


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

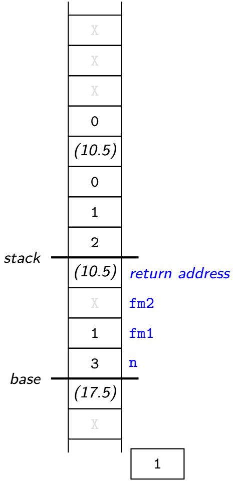


fib, körning

code/fib.c

```
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

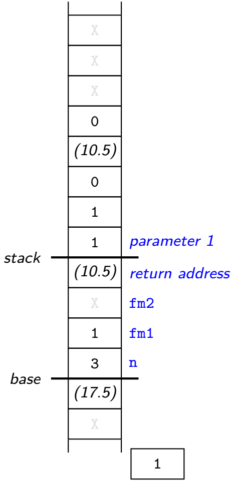


fib, körning

code/fib.c

```
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       int fm1, fm2;
6       if (n < 2) {
7           return n;
8       } else {
9           fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

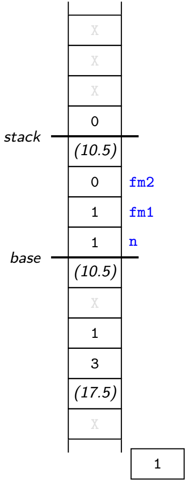


fib, körning

code/fib.c

```c
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

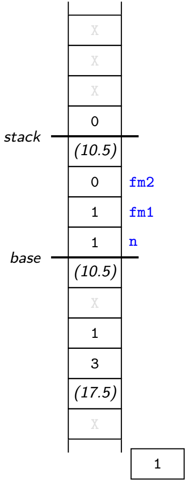


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

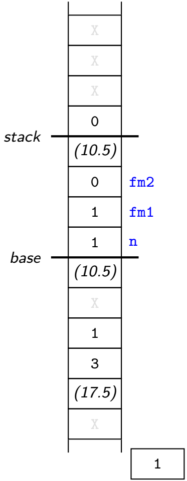


fib, körning

code/fib.c

```
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]



	X	
	X	
	X	
	0	
stack		
	(10.5)	
	0	fm2
	1	fm1
	1	n
base		
	(10.5)	
	X	
	1	
	3	
	(17.5)	
	X	

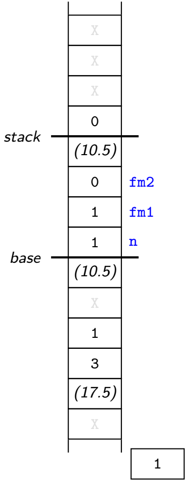
1

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

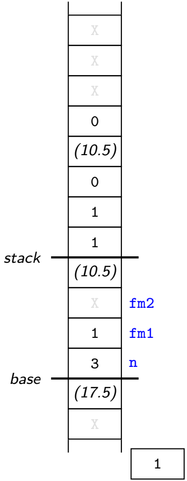


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

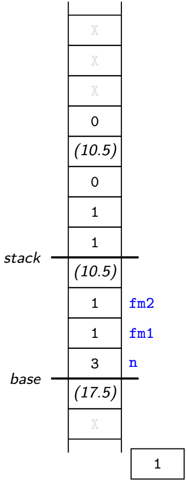


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

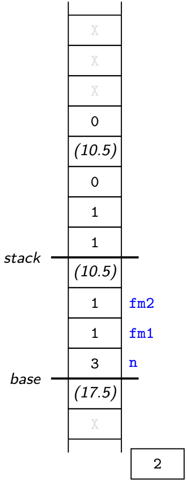


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

[redacted]

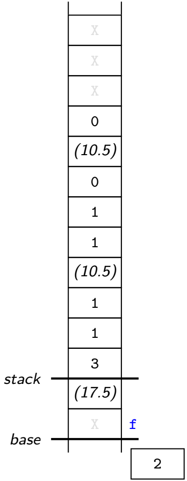


fib, körning

code/fib.c

```c
 1  #include <stdio.h>
 2
 3  int fib(int n)
 4  {
 5      int fm1, fm2;
 6      if (n < 2) {
 7          return n;
 8      } else {
 9          fm1 = fib(n - 1);
10          fm2 = fib(n - 2);
11          return fm1 + fm2;
12      }
13  }
14
15  int main(void)
16  {
17      int f = fib(3);
18      printf("fib(3) = %d\n", f);
19      return 0;
20  }
```

[redacted]

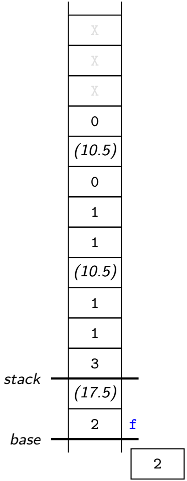


fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

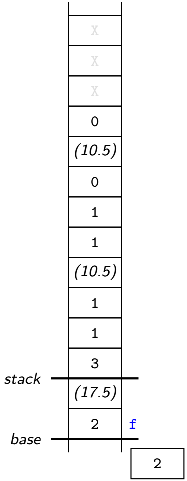
[redacted]



fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



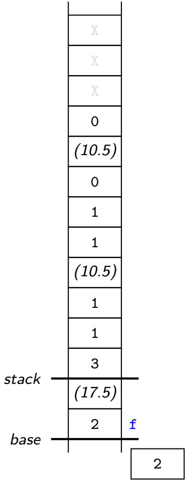
```
fib(3) = 2
```

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

```
fib(3) = 2
```

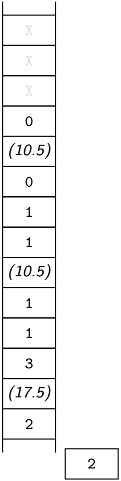


X
X
X
0
(10.5)
0
1
1
(10.5)
1
1
3
stack
(17.5)
2 f
base
2

fib, körning

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



```
fib(3) = 2
```

fib, anropsträd

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

main

fib, anropsträd

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```

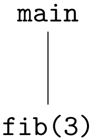
main

fib, anropsträd

3

code/fib.c

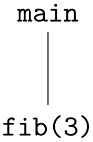
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

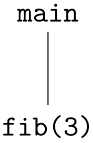
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



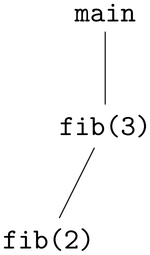
main

fib(3)

fib, anropsträd

code/fib.c

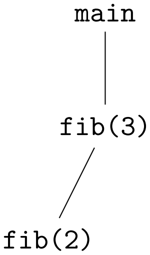
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



main

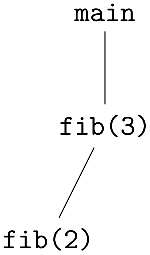
fib(3)

fib(2)

fib, anropsträd

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



main

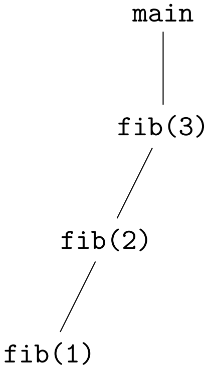
fib(3)

fib(2)

fib, anropsträd

code/fib.c

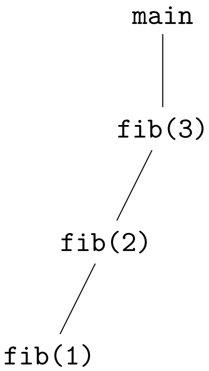
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

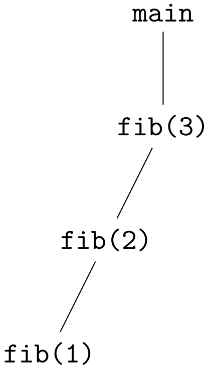
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

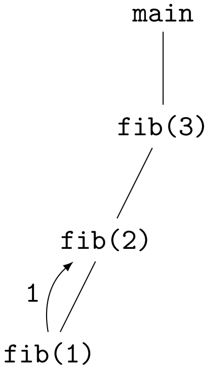
```
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

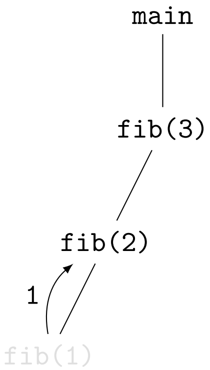
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

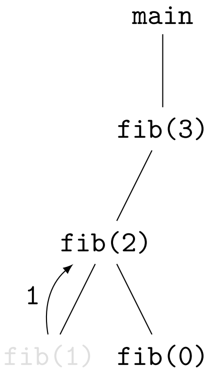
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

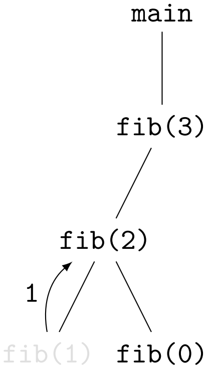
```
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

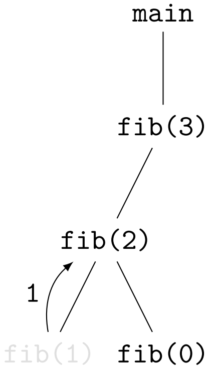
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

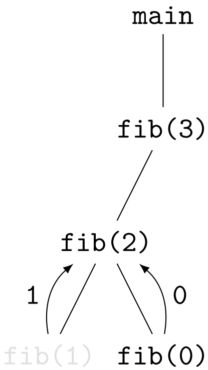
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

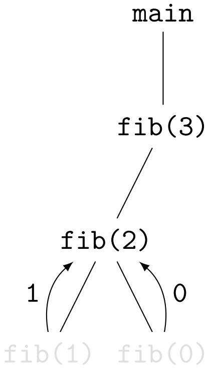
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

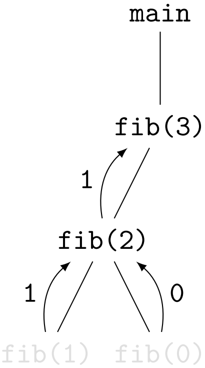
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



main

fib(3)

1

fib(2)

1

0

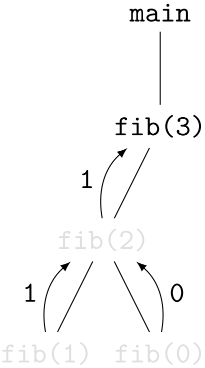
fib(1)

fib(0)

fib, anropsträd

code/fib.c

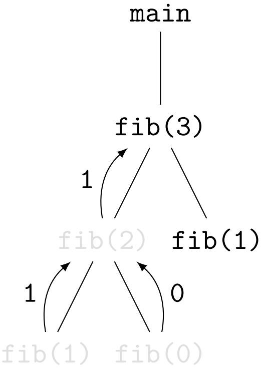
```
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

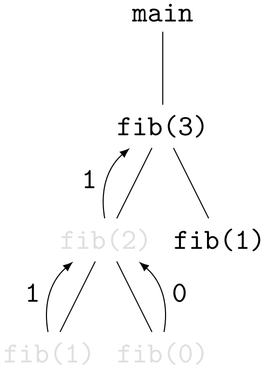
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

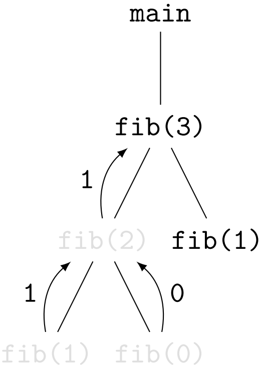
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

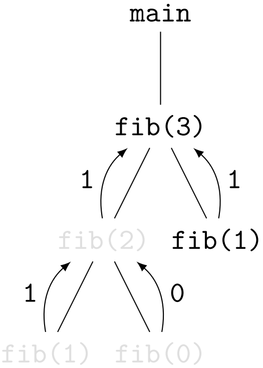
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

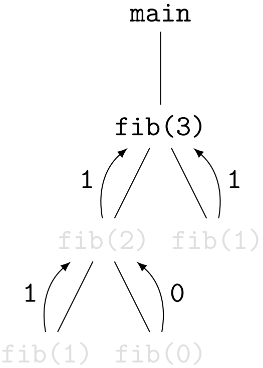
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

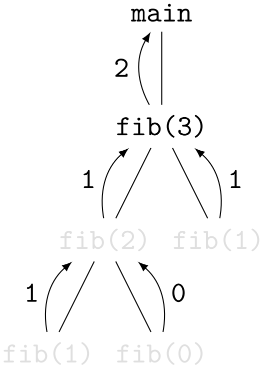
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

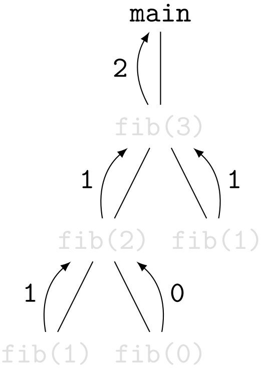
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

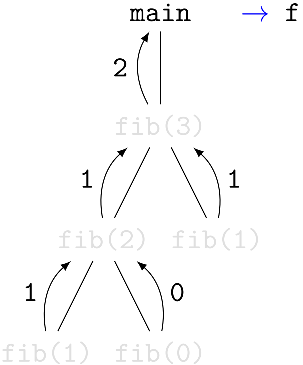
```
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

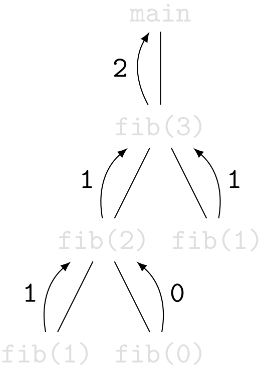
```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



fib, anropsträd

code/fib.c

```c
#include <stdio.h>

int fib(int n)
{
    int fm1, fm2;
    if (n < 2) {
        return n;
    } else {
        fm1 = fib(n - 1);
        fm2 = fib(n - 2);
        return fm1 + fm2;
    }
}

int main(void)
{
    int f = fib(3);
    printf("fib(3) = %d\n", f);
    return 0;
}
```



När är det lämpligt med rekursion?

- Rekursion är ofta en bra lösning om följande villkor är uppfyllda:
  1. Ett eller flera enkla fall har en enkel, icke-rekursiv lösning
     - Ex: tomt fält
  2. Mer komplicerade fall går att definiera i termer av fall som ligger närmare de enklaste fallen
     - Ex: ta hand om första halvan av fältet och sedan andra halvan av fältet
- Genom att tillämpa denna omdefiniering varje gång funktionen anropas reduceras problemet till slut helt och hållet till de enklaste fallen

Rekursion vs iteration

- Fördelar med rekursion är att
  - Det blir ofta eleganta, kompakta kodlösningar
  - Det lämpar sig oerhört väl för problem som är rekursiva i sin natur, tex att söka information i ett träd
- Nackdelar är
  - Ineffektivitet: Vid varje funktionsanrop skall en massa data sparas undan, etc.
  - Olämpligt för applikationer som skall iterera för alltid

Vanliga fel vid rekursion

- Det vanligaste problemet är att något basfall inte nås
  - Endera så saknas basfall. . .
  - . . . eller så tar inte det rekursiva fallet problemet närmare ett basfall

Binärsökning igen

- Algoritm med vanliga ord
  1. Jämför med elementet närmast mitten i sekvensen
     1.1 Om likhet — klart
     1.2 Om det sökta värdet kommer före elementet närmast mitten, sök i den vänstra delsekvensen, hoppa till steg 1
     1.3 Om det sökta värdet kommer efter elementet närmast mitten, sök i den högra delsekvensen, hoppa till steg 1
- Rekursiv!

Binärsökning - rekursiv

- Rekursiv algoritm:

```
// "Starter" function to be called from the outside
Algorithm binsearch(a: Array, n: Int, v: Value)
  return binsearch_rec(a, 0, n - 1, v)

// Internal recursive function, not visible from the outside
Algorithm binsearch_rec(a: Array, left, right: Int, val: Value)
  mid <- (left + right) / 2
  if left > right then
      return -1                                   // Not found
  else if val = a[mid] then
      return mid                                  // Found it
  else if val < a[mid] then                     // Look left
      return binsearch_rec(a, left, mid - 1, val)
  else // val > a[mid]                          // Look right
      return binsearch_rec(a, mid + 1, right, val)
```

Binärsökning - iterativ

- Iterativ algoritm igen:

```
Algorithm binsearch(a: Array, n: Int, v: Value)
  left <- 0
  right <- n - 1
  while left <= right do
    mid  <- (left + right) / 2   // Integer division
    if v = a[mid] then
      return mid                 // Found it
    else if v < a[mid] then
      right <-  mid - 1          // Look left
    else
      left <- mid + 1            // Look right

  return -1 // Not found
```

Sortering

Sortering

- Varför ska man sortera?
  - Snabba upp andra algoritmer genom att vi vet mer
    - Sökning
    - Hantera stora datamängder
- Det finns flera olika algoritmer för sortering
- Vi kommer att titta på tre olika
  - Instickssortering — *Insertion Sort*
  - Bubbel-sortering — *Bubble Sort*
  - Samsortering — *Merge Sort*
- Syfte:
  - Förstå principerna, känna igen algoritmerna
  - Behöver inte kunna implementera

Insertion sort av fält

- Algoritmen i grova drag:
    - Börja med ett element (ett element är sorterat)
    - Ta sedan ett element i taget och sortera in på rätt plats bland de tidigare sorterade elementen

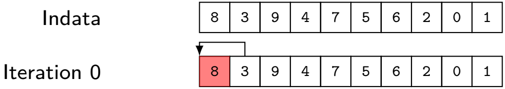
Insertion sort — exempel

Indata	8	3	9	4	7	5	6	2	0	1

Insertion sort — exempel

Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1

Insertion sort — exempel

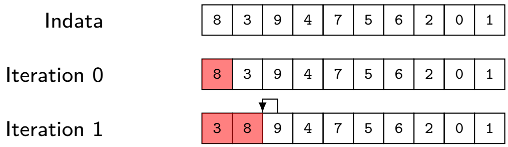


Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1

Insertion sort — exempel

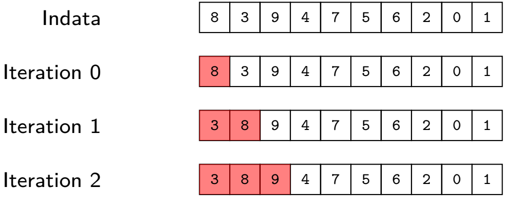
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	**8**	3	9	4	7	5	6	2	0	1
Iteration 1	**3**	**8**	9	4	7	5	6	2	0	1

Insertion sort — exempel



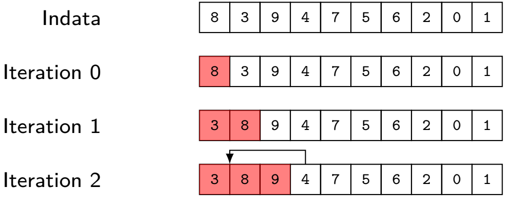
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1

Insertion sort — exempel



Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1

Insertion sort — exempel

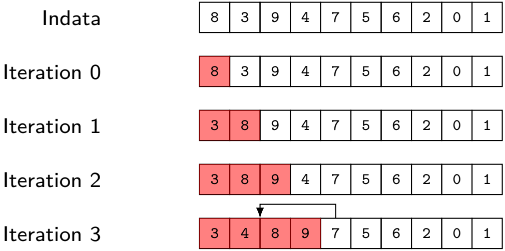


Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1

Insertion sort — exempel

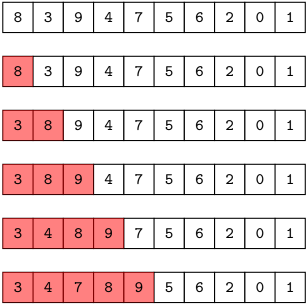
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	**8**	3	9	4	7	5	6	2	0	1
Iteration 1	**3**	**8**	9	4	7	5	6	2	0	1
Iteration 2	**3**	**8**	**9**	4	7	5	6	2	0	1
Iteration 3	**3**	**4**	**8**	**9**	7	5	6	2	0	1

Insertion sort — exempel



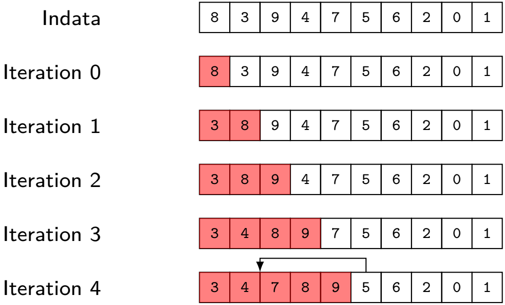
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1

Insertion sort — exempel



Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1

Insertion sort — exempel

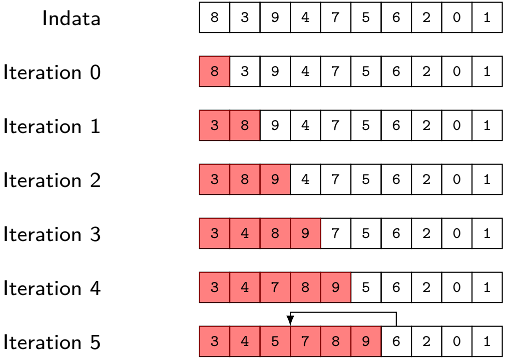


Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1

Insertion sort — exempel

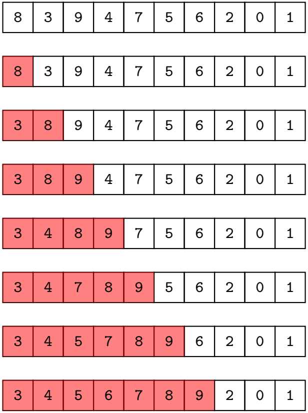
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	**8**	3	9	4	7	5	6	2	0	1
Iteration 1	**3**	**8**	9	4	7	5	6	2	0	1
Iteration 2	**3**	**8**	**9**	4	7	5	6	2	0	1
Iteration 3	**3**	**4**	**8**	**9**	7	5	6	2	0	1
Iteration 4	**3**	**4**	**7**	**8**	**9**	5	6	2	0	1
Iteration 5	**3**	**4**	**5**	**7**	**8**	**9**	6	2	0	1

Insertion sort — exempel



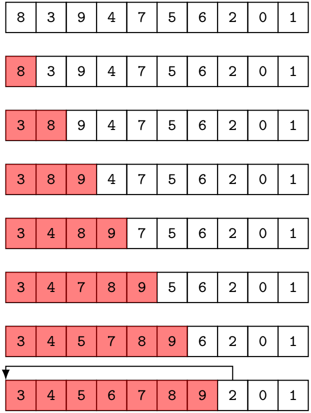
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1

Insertion sort — exempel



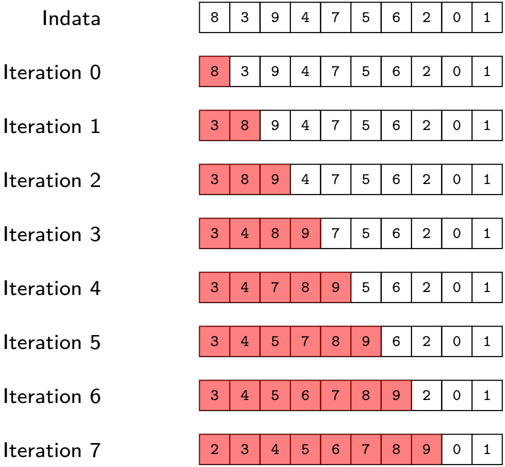
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1
Iteration 6	3	4	5	6	7	8	9	2	0	1

Insertion sort — exempel



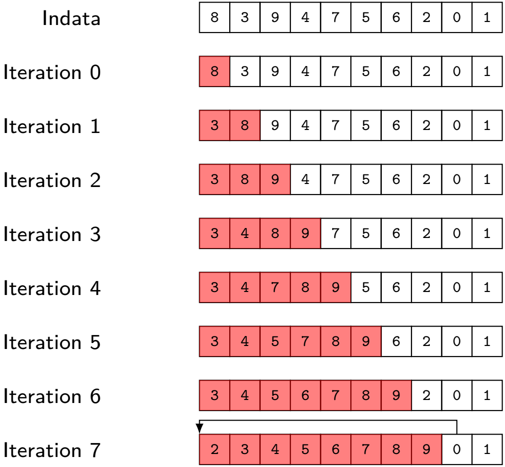
Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1
Iteration 6	3	4	5	6	7	8	9	2	0	1

Insertion sort — exempel



Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1
Iteration 6	3	4	5	6	7	8	9	2	0	1
Iteration 7	2	3	4	5	6	7	8	9	0	1

Insertion sort — exempel

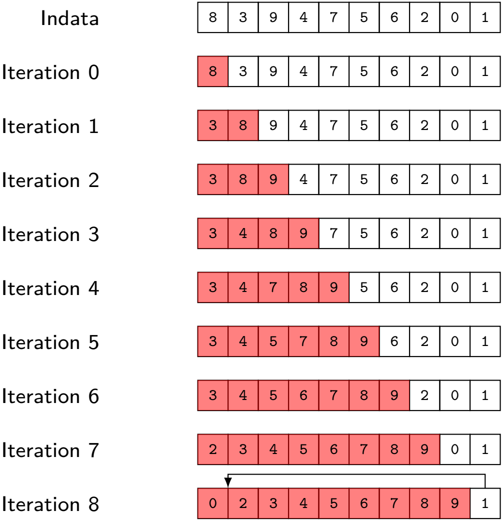


Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1
Iteration 6	3	4	5	6	7	8	9	2	0	1
Iteration 7	2	3	4	5	6	7	8	9	0	1

Insertion sort — exempel

Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1
Iteration 6	3	4	5	6	7	8	9	2	0	1
Iteration 7	2	3	4	5	6	7	8	9	0	1
Iteration 8	0	2	3	4	5	6	7	8	9	1

Insertion sort — exempel

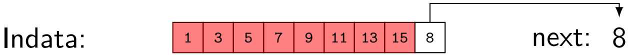


Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1
Iteration 6	3	4	5	6	7	8	9	2	0	1
Iteration 7	2	3	4	5	6	7	8	9	0	1
Iteration 8	0	2	3	4	5	6	7	8	9	1

Insertion sort — exempel

Indata	8	3	9	4	7	5	6	2	0	1
Iteration 0	8	3	9	4	7	5	6	2	0	1
Iteration 1	3	8	9	4	7	5	6	2	0	1
Iteration 2	3	8	9	4	7	5	6	2	0	1
Iteration 3	3	4	8	9	7	5	6	2	0	1
Iteration 4	3	4	7	8	9	5	6	2	0	1
Iteration 5	3	4	5	7	8	9	6	2	0	1
Iteration 6	3	4	5	6	7	8	9	2	0	1
Iteration 7	2	3	4	5	6	7	8	9	0	1
Iteration 8	0	2	3	4	5	6	7	8	9	1
Utdata	0	1	2	3	4	5	6	7	8	9

Insertion sort — Sidospår: insättning (1)



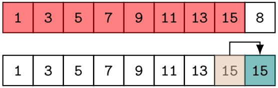
Indata:

1	3	5	7	9	11	13	15	8

next: 8

Insertion sort — Sidospår: insättning (1)

Indata:



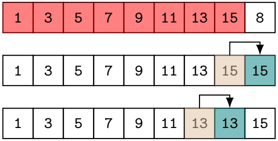
1	3	5	7	9	11	13	15	8

1	3	5	7	9	11	13	15	15

next: 8

Insertion sort — Sidospår: insättning (1)

Indata:



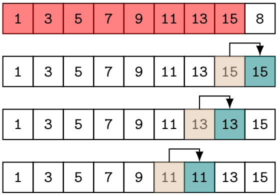
1	3	5	7	9	11	13	15	8
1	3	5	7	9	11	13	15	15
1	3	5	7	9	11	13	13	15

next: 8

Insertion sort — Sidospår: insättning (1)

Indata:

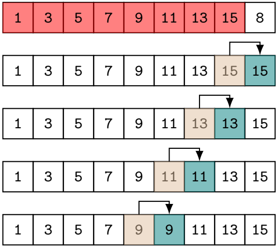
next: 8



1	3	5	7	9	11	13	15	8
1	3	5	7	9	11	13	15	15
1	3	5	7	9	11	13	13	15
1	3	5	7	9	11	11	13	15

Insertion sort — Sidospår: insättning (1)

Indata:



1 3 5 7 9 11 13 15 8

1 3 5 7 9 11 13 15 15

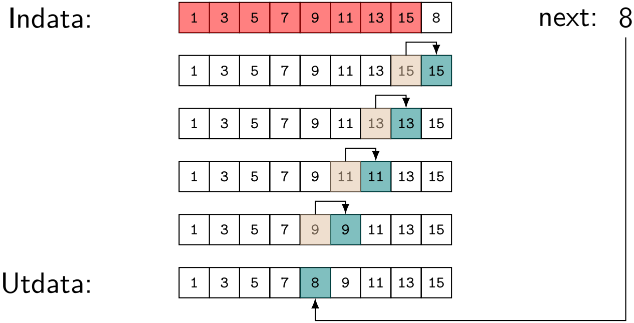
1 3 5 7 9 11 13 13 15

1 3 5 7 9 11 11 13 15

1 3 5 7 9 9 11 13 15

next: 8

Insertion sort — Sidospår: insättning (1)



Indata: 1 3 5 7 9 11 13 15 8   next: 8

1 3 5 7 9 11 13 15 15

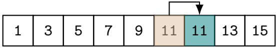
1 3 5 7 9 11 13 13 15

1 3 5 7 9 11 11 13 15

1 3 5 7 9 9 11 13 15

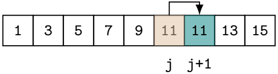
Utdata: 1 3 5 7 8 9 11 13 15

Insertion sort — Sidospår: insättning (2)



1	3	5	7	9	11	11	13	15

Insertion sort — Sidospår: insättning (2)



1	3	5	7	9	11	11	13	15
					j	j+1		

Insertion sort — algoritm

- Algoritm:

```
Algorithm insertion_sort(a: Array, n: Int)
  // i indicates first unsorted element in a

  for i <- 1 to n - 1 do

    // new value to insert in sorted part of a
    next <- a[i]

    // start with last sorted element
    j <- i - 1

    // as long as new element is smaller and
    // we're inside the array
    while j >= 0 and next < a[j] do

      // shift element right
      a[j + 1] <- a[j]

      // continue to the left
      j <- j - 1

    // insert new value in its sorted place
    a[j+1] <- next

  return a
```

Bubble Sort

- Algoritmen i grova drag:
    - Upprepa följande tills ingen förändring sker:
        - Jämför alla elementen ett par i taget
            - Börja med element 0 och 1, därefter 1 och 2, osv
        - Om elementen är i fel ordning, byt plats på dem

Bubble Sort — algoritm

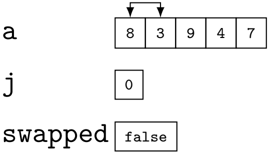
- Algoritm:

```
Algorithm bubble_sort(a: Array, n: Int)
  do
    // so far no swap has taken place
    swapped <- false

    // for each adjacent pair in a...
    for j <- 0 to n - 2 do

      // if the elements are in the wrong order...
      if a[j] > a[j + 1] then

        // ...swap the elements
        tmp <- a[j]
        a[j] <- a[j + 1]
        a[j + 1] <- tmp

        // remember that a swap has taken place
        swapped <- true

  while swapped = true

  return a
```

Bubble Sort exempel

a	8	3	9	4	7
j	X				
swapped	X				

Bubble Sort exempel

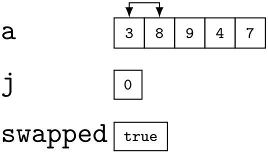


a
8 3 9 4 7

j
0

swapped
false

Bubble Sort exempel

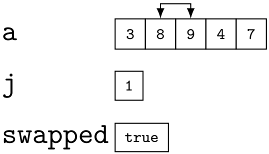


a | 3 | 8 | 9 | 4 | 7

j | 0

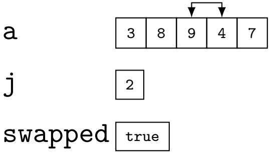
swapped | true

Bubble Sort exempel



a	3	8	9	4	7
j	1				
swapped	true				

Bubble Sort exempel

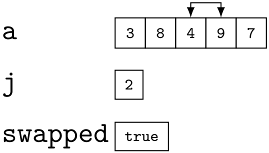


a 3 8 9 4 7

j 2

swapped true

Bubble Sort exempel

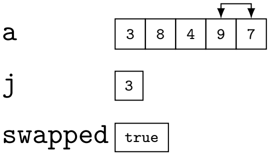


a 3 8 4 9 7

j 2

swapped true

Bubble Sort exempel

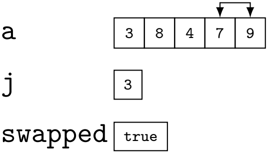


a | 3 | 8 | 4 | 9 | 7

j | 3

swapped | true

Bubble Sort exempel

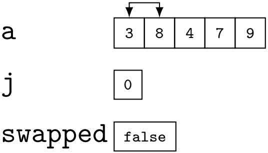


a 3 8 4 7 9

j 3

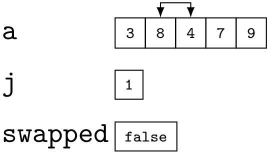
swapped true

Bubble Sort exempel



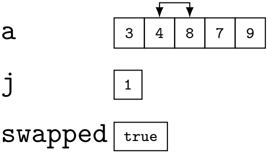
a	3	8	4	7	9
j	0				
swapped	false				

Bubble Sort exempel



a	3	8	4	7	9
j	1				
swapped	false				

Bubble Sort exempel



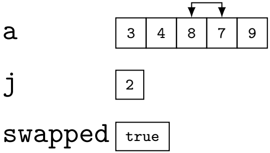
a

3	4	8	7	9

j 1

swapped true

Bubble Sort exempel

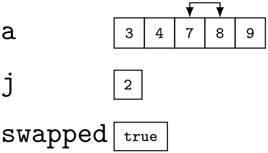


a | 3 | 4 | 8 | 7 | 9

j | 2

swapped | true

Bubble Sort exempel

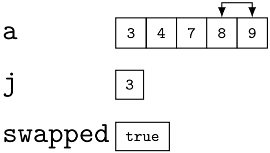


a 3 4 7 8 9

j 2

swapped true

Bubble Sort exempel

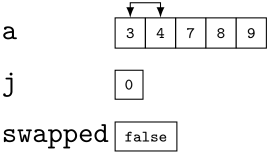


a 3 4 7 8 9

j 3

swapped true

Bubble Sort exempel

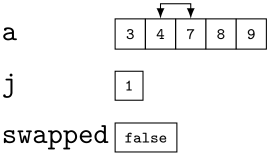


a 3 4 7 8 9

j 0

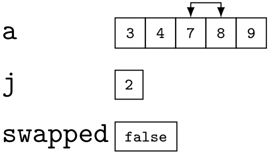
swapped false

Bubble Sort exempel

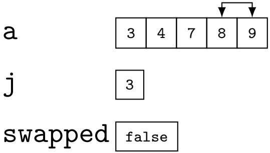


a	3	4	7	8	9
j	1				
swapped	false				

Bubble Sort exempel

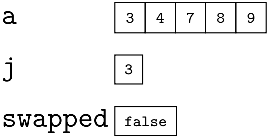


Bubble Sort exempel



```
a        3 4 7 8 9
j        3
swapped  false
```

Bubble Sort exempel



a 3 4 7 8 9

j 3

swapped false

Merge Sort

- Algoritmen i grova drag
  - Om sekvensen har ett element
    - Returnera sekvensen (den är redan sorterad)
  - annars
    - Dela sekvensen i två ungefär lika stora delsekvenser
    - Sortera delsekvenserna rekursivt
    - Slå samman delsekvenserna (*Merge*)
    - Returnera den sammanslagna sekvensen

Merge

- *Merge Sort* använder en delalgoritm — *Merge*
- Algoritm för att slå samman två redan sorterade sekvenser:
  - Så länge bägge sekvenserna har element:
    - Jämför första (=minsta) elementet i vardera sekvensen
    - Flytta det minsta av de två elementen till utsekvensen
  - Flytta över alla element som finns kvar i sekvenserna

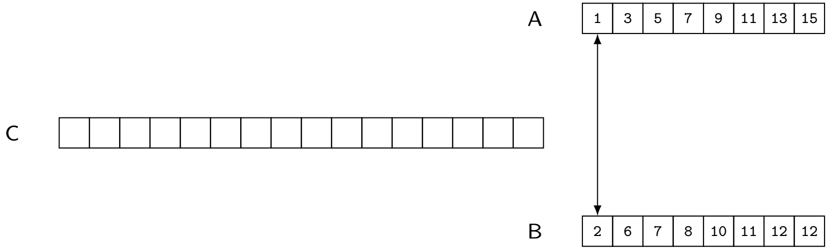
Merge — exempel

A	1	3	5	7	9	11	13	15

C																

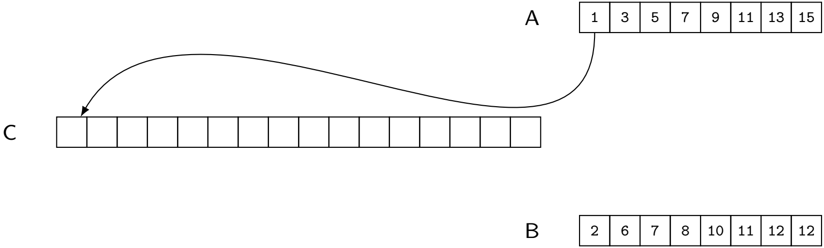
B	2	6	7	8	10	11	12	12

Merge — exempel



A	1	3	5	7	9	11	13	15
B	2	6	7	8	10	11	12	12

Merge — exempel

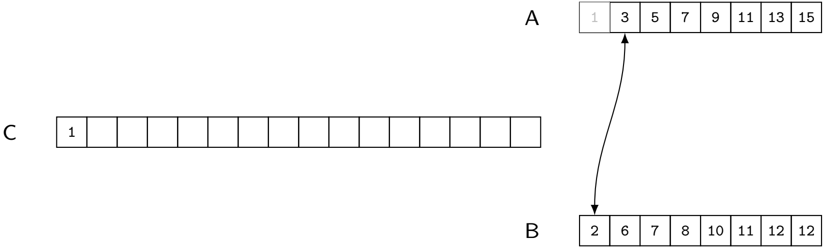


A	1	3	5	7	9	11	13	15

C

B	2	6	7	8	10	11	12	12

Merge — exempel

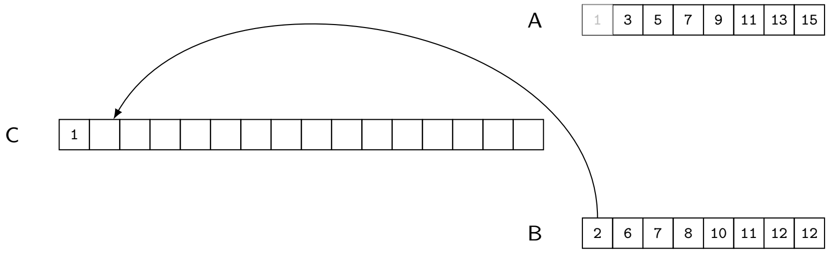


A	1	3	5	7	9	11	13	15

C	1															

B	2	6	7	8	10	11	12	12

Merge — exempel



A

1	3	5	7	9	11	13	15

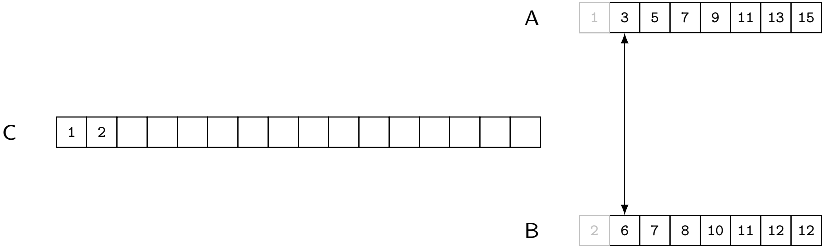
C

1															

B

2	6	7	8	10	11	12	12

Merge — exempel

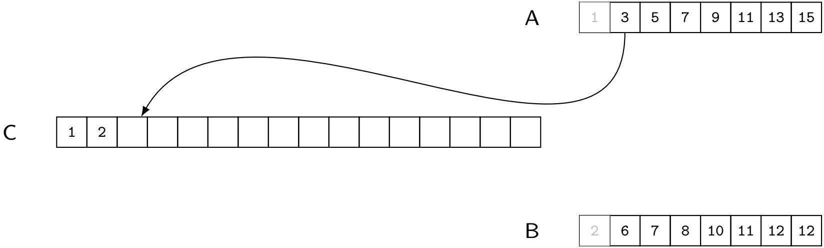


A: 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15

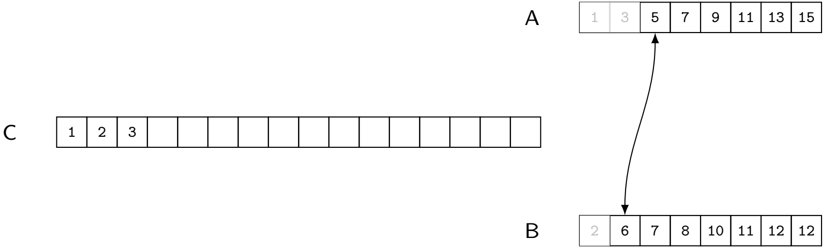
C: 1 | 2 | | | | | | | | | | | | | |

B: 2 | 6 | 7 | 8 | 10 | 11 | 12 | 12

Merge — exempel

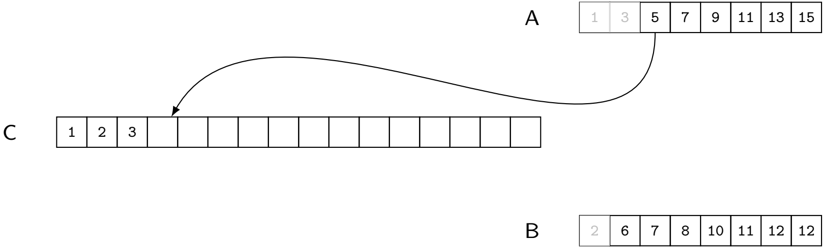


Merge — exempel



A	1	3	5	7	9	11	13	15								
C	1	2	3													
B	2	6	7	8	10	11	12	12								

Merge — exempel

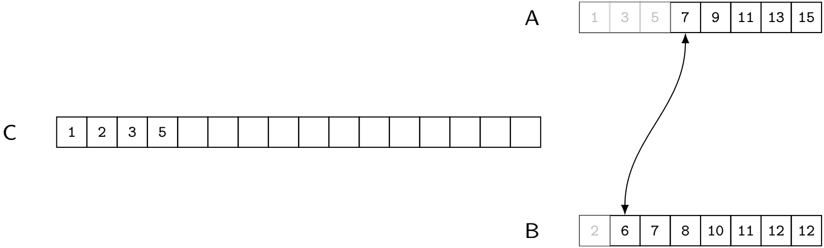


A							
1	3	5	7	9	11	13	15

C															
1	2	3													

B							
2	6	7	8	10	11	12	12

Merge — exempel

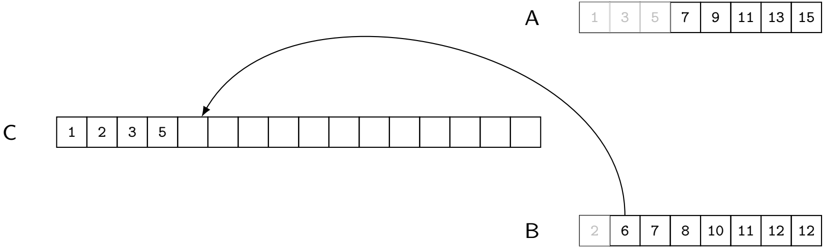


A	1	3	5	7	9	11	13	15

C	1	2	3	5												

B	2	6	7	8	10	11	12	12

Merge — exempel

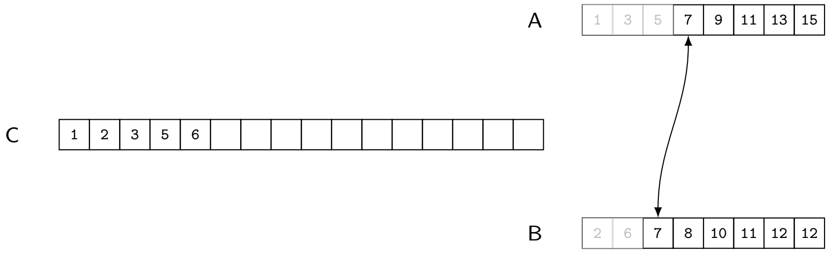


A: 1 3 5 7 9 11 13 15

C: 1 2 3 5

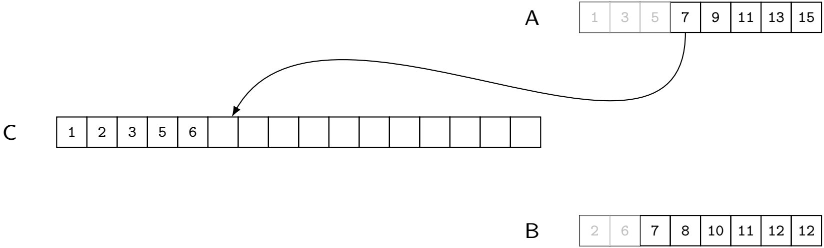
B: 2 6 7 8 10 11 12 12

Merge — exempel



A	1	3	5	7	9	11	13	15								
C	1	2	3	5	6											
B	2	6	7	8	10	11	12	12								

Merge — exempel

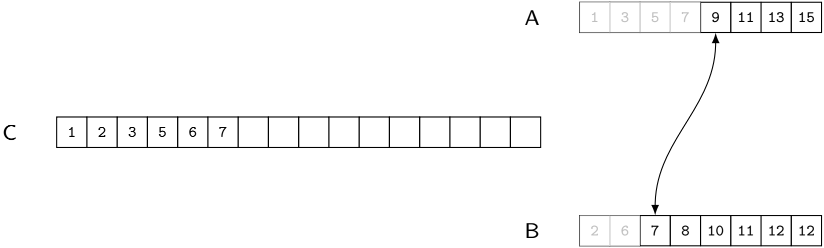


A							
1	3	5	7	9	11	13	15

C															
1	2	3	5	6											

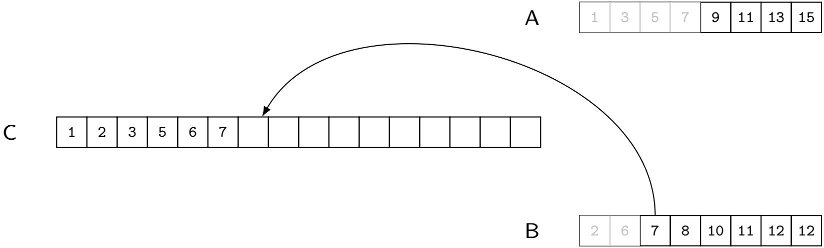
B							
2	6	7	8	10	11	12	12

Merge — exempel



A	1	3	5	7	9	11	13	15								
C	1	2	3	5	6	7										
B	2	6	7	8	10	11	12	12								

Merge — exempel



A

1	3	5	7	9	11	13	15

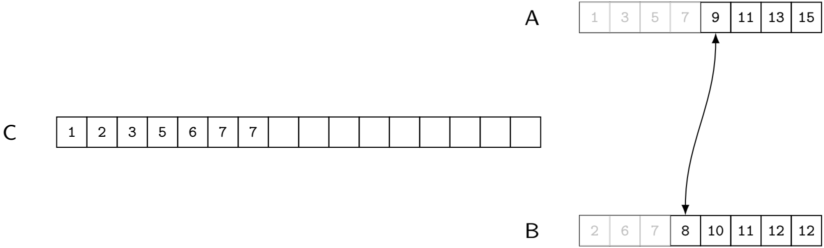
C

1	2	3	5	6	7										

B

2	6	7	8	10	11	12	12

Merge — exempel

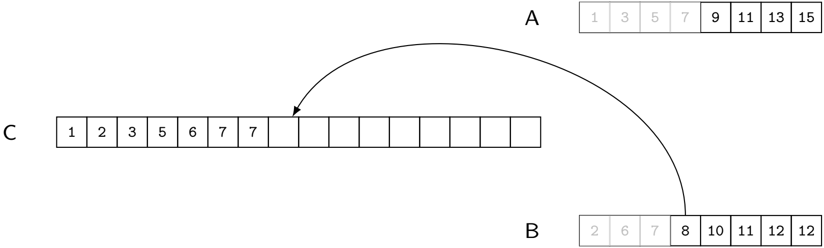


A: 1 3 5 7 9 11 13 15

C: 1 2 3 5 6 7 7

B: 2 6 7 8 10 11 12 12

Merge — exempel

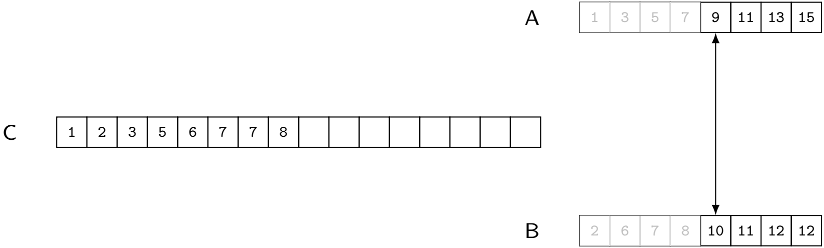


A: 1 3 5 7 9 11 13 15

C: 1 2 3 5 6 7 7

B: 2 6 7 8 10 11 12 12

Merge — exempel

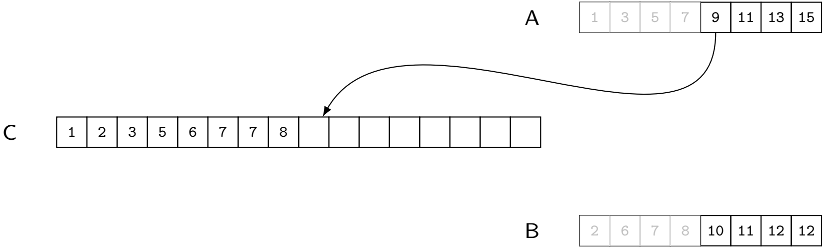


A 1 3 5 7 9 11 13 15

C 1 2 3 5 6 7 7 8

B 2 6 7 8 10 11 12 12

Merge — exempel

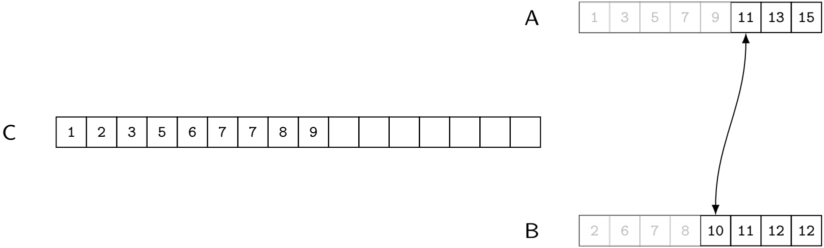


A 1 3 5 7 9 11 13 15

C 1 2 3 5 6 7 7 8

B 2 6 7 8 10 11 12 12

Merge — exempel



A

1	3	5	7	9	11	13	15

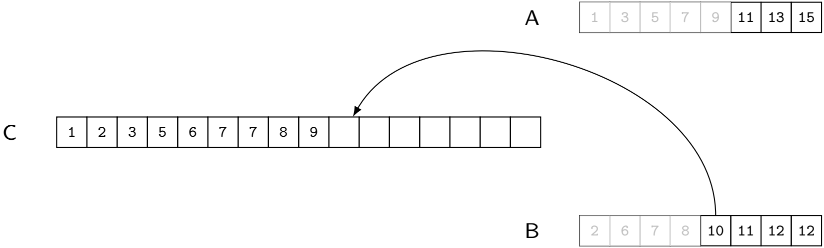
C

1	2	3	5	6	7	7	8	9							

B

2	6	7	8	10	11	12	12

Merge — exempel

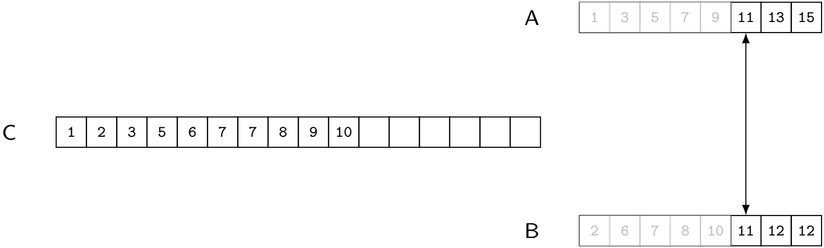


A	1	3	5	7	9	11	13	15

C	1	2	3	5	6	7	7	8	9							

B	2	6	7	8	10	11	12	12

Merge — exempel

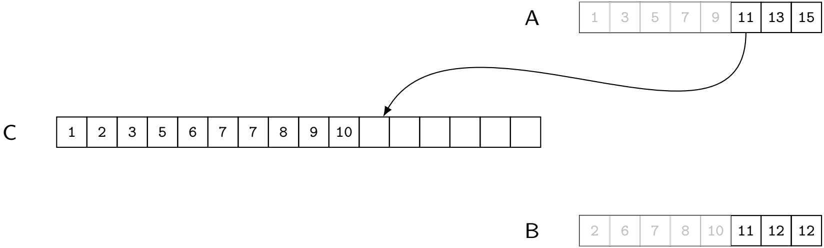


A | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15

C | 1 | 2 | 3 | 5 | 6 | 7 | 7 | 8 | 9 | 10 | | | | | |

B | 2 | 6 | 7 | 8 | 10 | 11 | 12 | 12

Merge — exempel

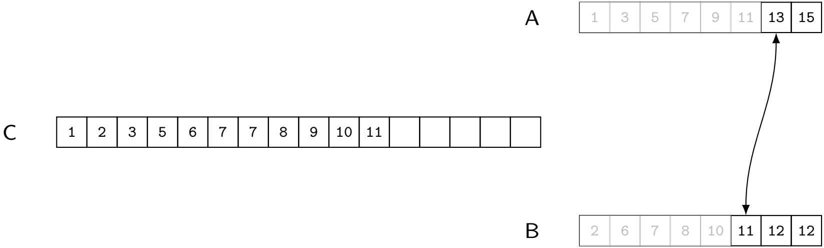


A: 1 3 5 7 9 11 13 15

C: 1 2 3 5 6 7 7 8 9 10

B: 2 6 7 8 10 11 12 12

Merge — exempel

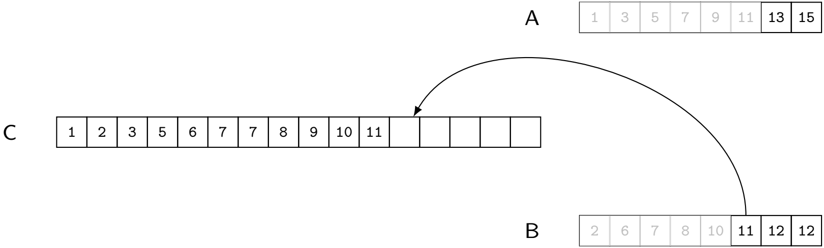


A: 1 3 5 7 9 11 13 15

C: 1 2 3 5 6 7 7 8 9 10 11

B: 2 6 7 8 10 11 12 12

Merge — exempel

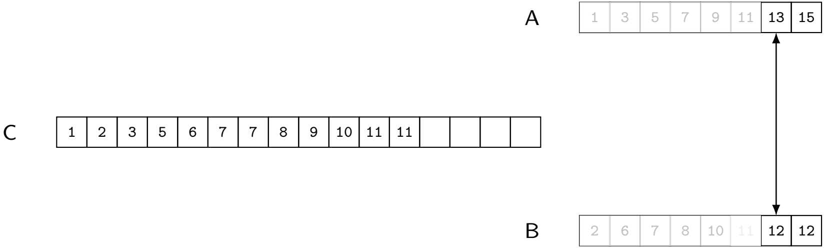


A 1 3 5 7 9 11 13 15

C 1 2 3 5 6 7 7 8 9 10 11

B 2 6 7 8 10 11 12 12

Merge — exempel

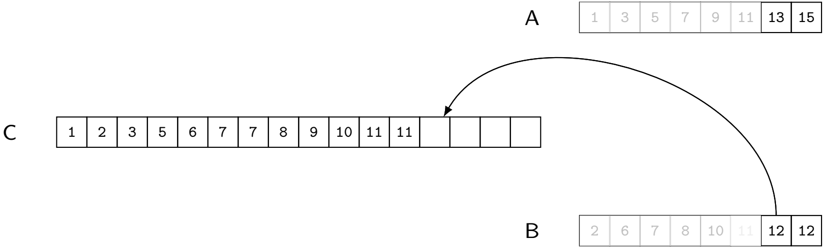


A								
	1	3	5	7	9	11	13	15

C																
	1	2	3	5	6	7	7	8	9	10	11	11				

B								
	2	6	7	8	10	11	12	12

Merge — exempel



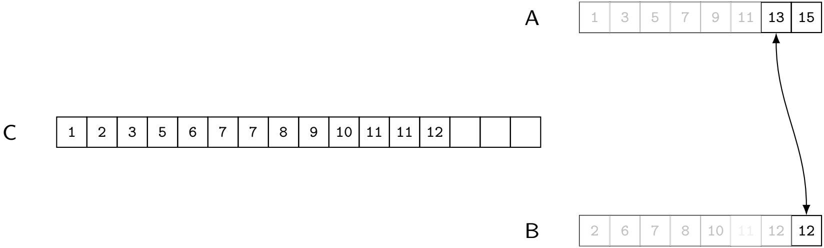
A																

A	1	3	5	7	9	11	13	15

C	1	2	3	5	6	7	7	8	9	10	11	11				

B	2	6	7	8	10	11	12	12

Merge — exempel

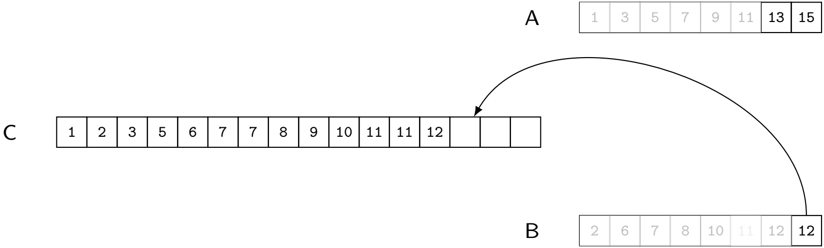


A								
	1	3	5	7	9	11	13	15

C																
	1	2	3	5	6	7	7	8	9	10	11	11	12			

B								
	2	6	7	8	10	11	12	12

Merge — exempel

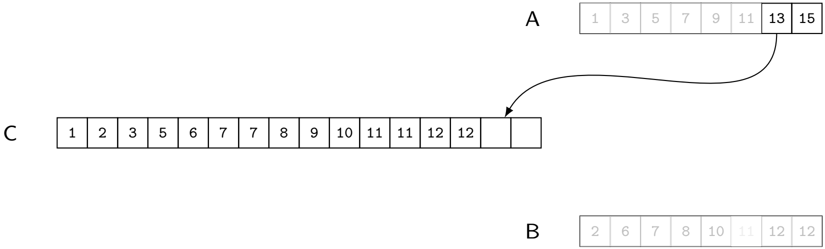


A: 1 3 5 7 9 11 13 15

C: 1 2 3 5 6 7 7 8 9 10 11 11 12

B: 2 6 7 8 10 11 12 12

Merge — exempel



A

1	3	5	7	9	11	13	15

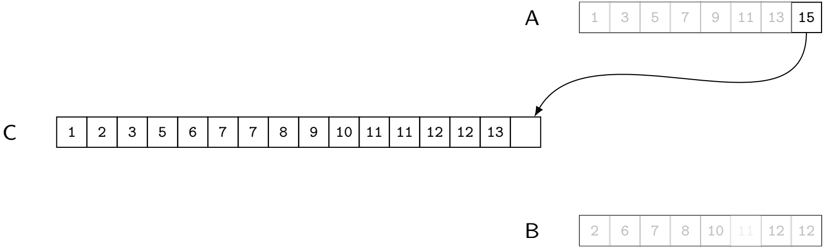
C

1	2	3	5	6	7	7	8	9	10	11	11	12	12		

B

2	6	7	8	10	11	12	12

Merge — exempel



A

1	3	5	7	9	11	13	15

C

1	2	3	5	6	7	7	8	9	10	11	11	12	12	13	

B

2	6	7	8	10	11	12	12

Merge — exempel

A

1	3	5	7	9	11	13	15

C

1	2	3	5	6	7	7	8	9	10	11	11	12	12	13	15

B

2	6	7	8	10	11	12	12

Merge

- Algoritm för *Merge*:

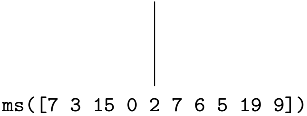
```
Algorithm merge(A, B: Array, na, nb: Int)
    C <- create_array(na + nb)

    ia <- 0 // Where to read from in A
    ib <- 0 // Where to read from in B
    ic <- 0 // Where to write to in C

    // While there are elements in both A and B...
    while ia < na and ib < nb do
        if A[ia] <= B[ib] then    // Smallest in A...
            C[ic] <- A[ia]        // ...copy from A
            ia <- ia + 1          // ...advance in A
        else                      // Smallest in B...
            C[ic] <- B[ib]        // ...copy from B
            ib <- ib + 1          // ...advance in B

        ic <- ic + 1              // Advance in C

    // While there are elements in A...
    while ia < na do
        C[ic] <- A[ia]            // ...copy from A
        ia <- ia + 1              // ...advance in A and C
        ic <- ic + 1

    // While there are elements in B...
    while ib < nb do
        C[ic] <- B[ib]            // ...copy from B
        ib <- ib + 1              // ...advance in B and C
        ic <- ic + 1

    return C
```

Merge Sort — algoritm

- Algoritm för *Merge Sort*:

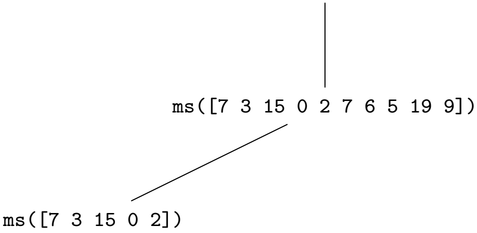
```
Algorithm merge_sort(a: Array, n: Int)
  if n < 2 then
    // Already sorted
    return a

  // Split a in two parts
  (left, right) <- split(a, n/2)

  // Lengths of left and right parts, respectively
  nl <- floor(n/2)
  nr <- n - nl

  // Sort left half recursively
  left <- merge_sort(left, nl)

  // Sort right half recursively
  right <- merge_sort(right, nr)

  // Merge sorted arrays
  a <- merge(left, right, nl, nr)

  return a
```

merge sort, anropsträd



ms([7 3 15 0 2 7 6 5 19 9])

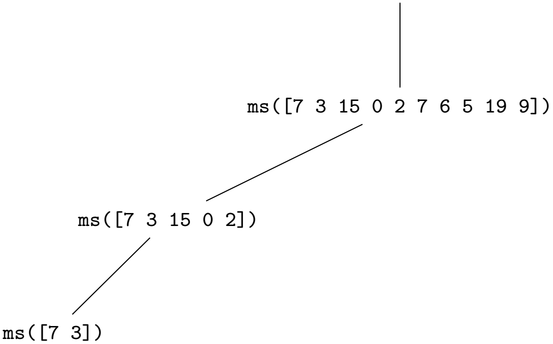
merge sort, anropsträd



ms([7 3 15 0 2 7 6 5 19 9])

ms([7 3 15 0 2])

`merge sort`, anropsträd

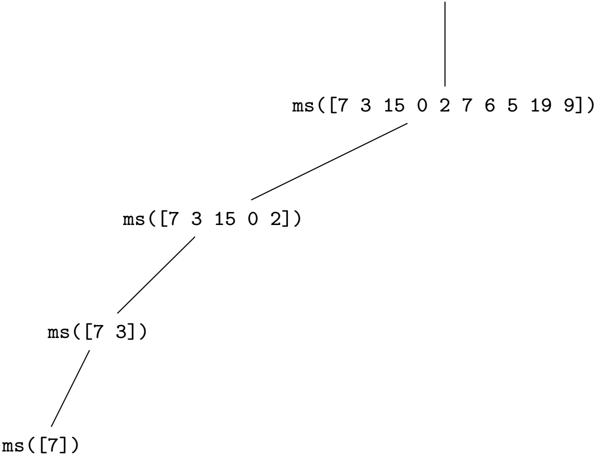


`ms([7 3 15 0 2 7 6 5 19 9])`

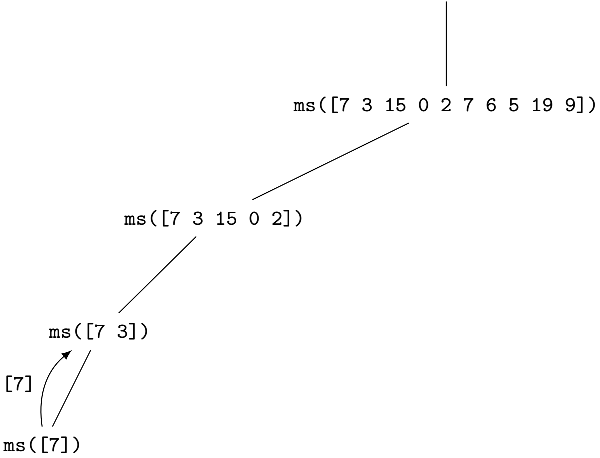
`ms([7 3 15 0 2])`

`ms([7 3])`

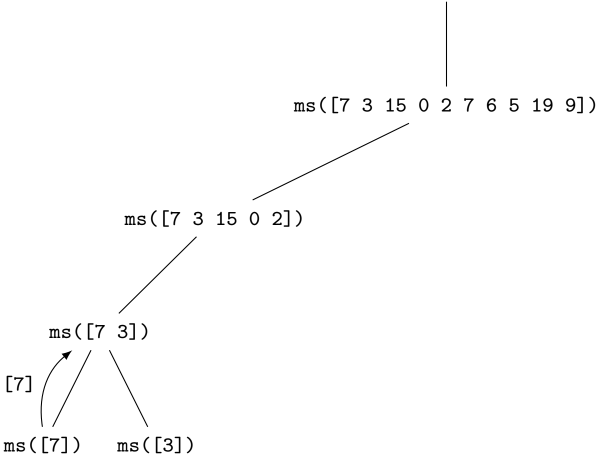
merge sort, anropsträd



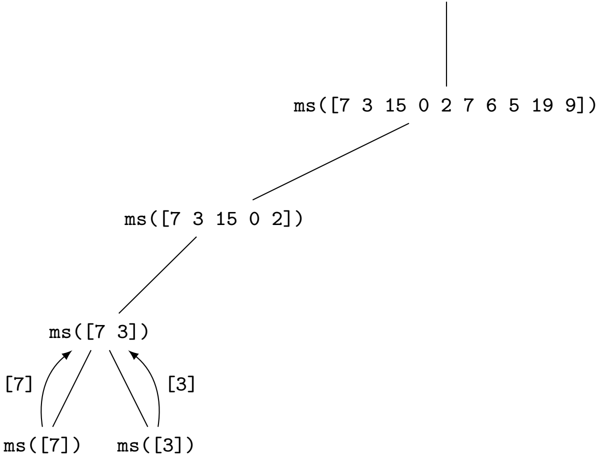
merge sort, anropsträd



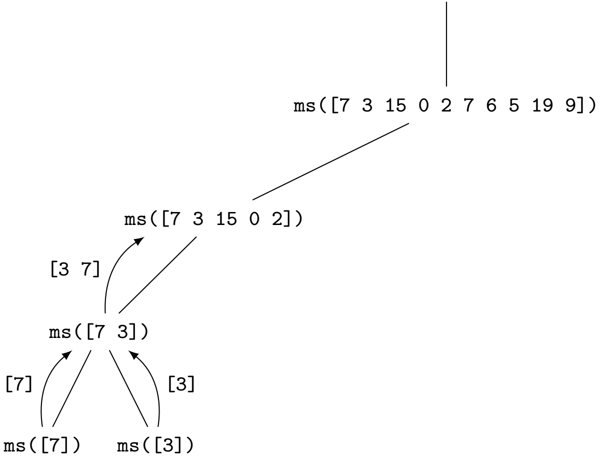
merge sort, anropsträd



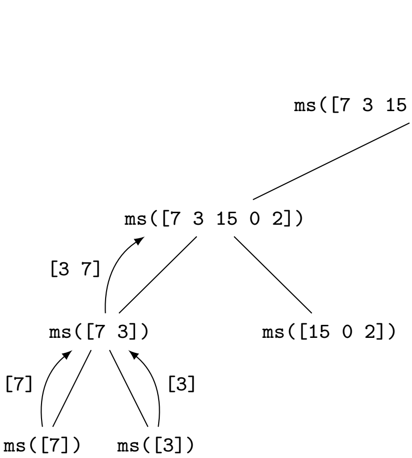
merge sort, anropsträd



merge sort, anropsträd



merge sort, anropsträd



ms([7 3 15 0 2 7 6 5 19 9])

ms([7 3 15 0 2])

[3 7]

ms([7 3])

ms([15 0 2])

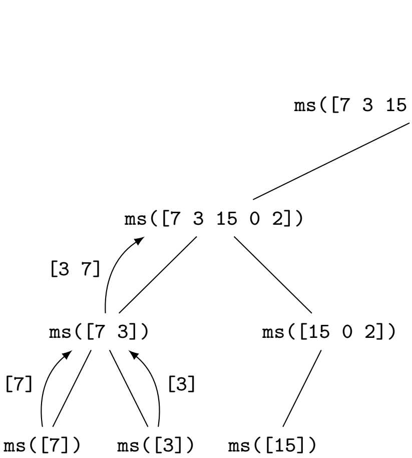
[7]

[3]

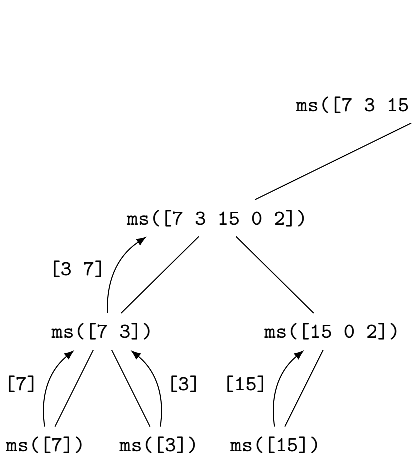
ms([7])

ms([3])

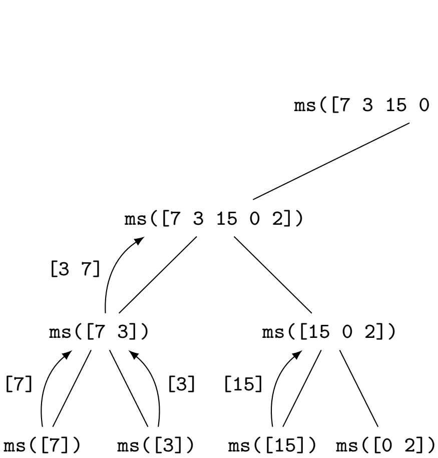
merge sort, anropsträd



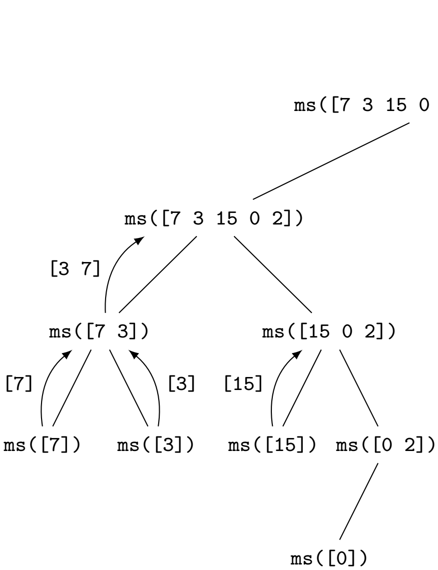
merge sort, anropsträd



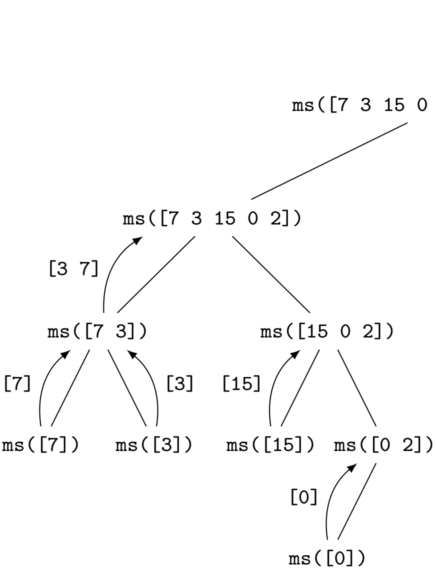
merge sort, anropsträd



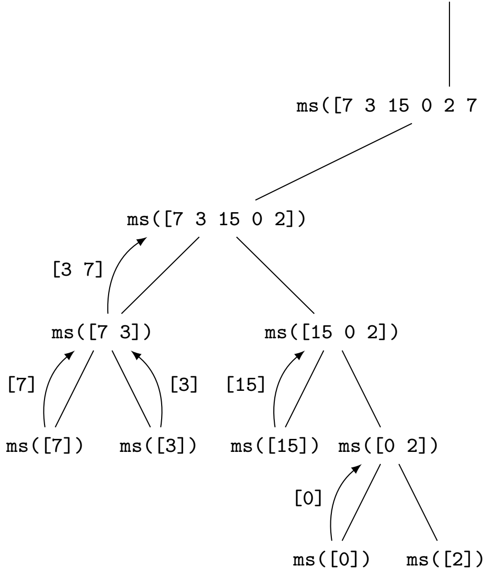
merge sort, anropsträd



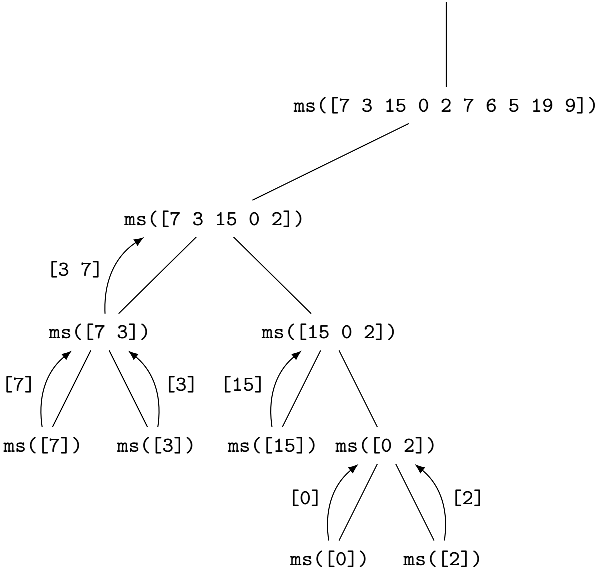
merge sort, anropsträd



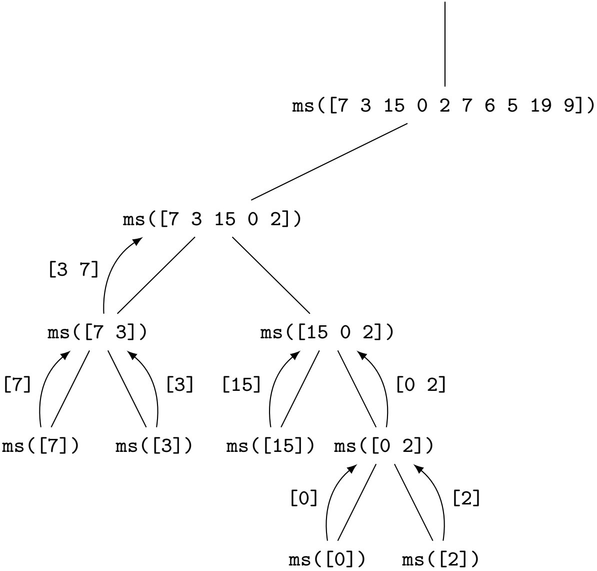
merge sort, anropsträd



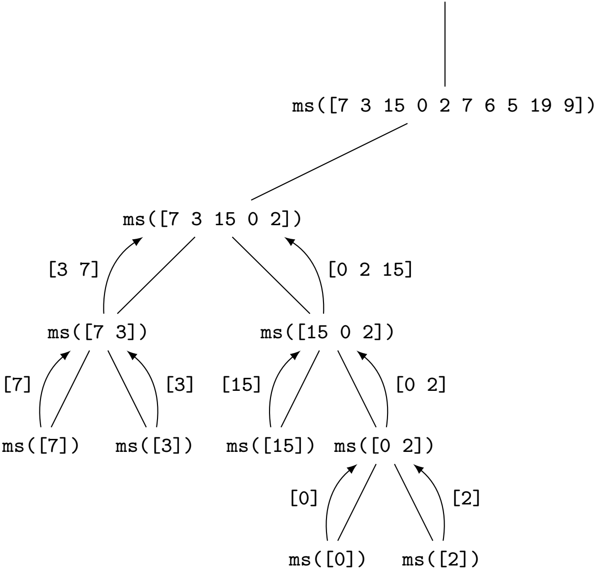
merge sort, anropsträd



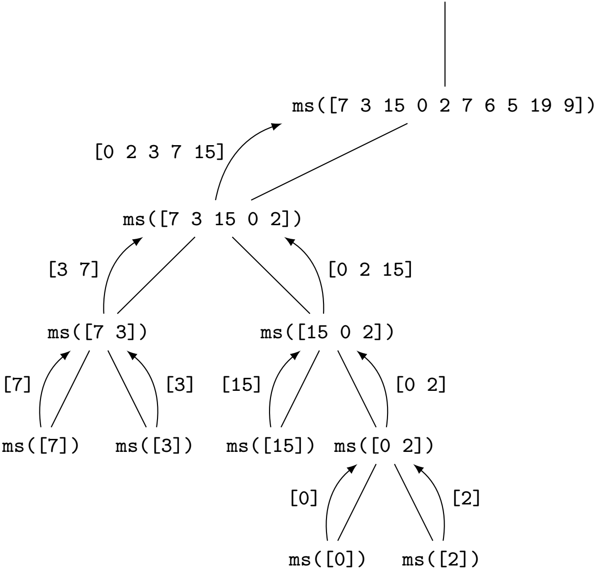
merge sort, anropsträd



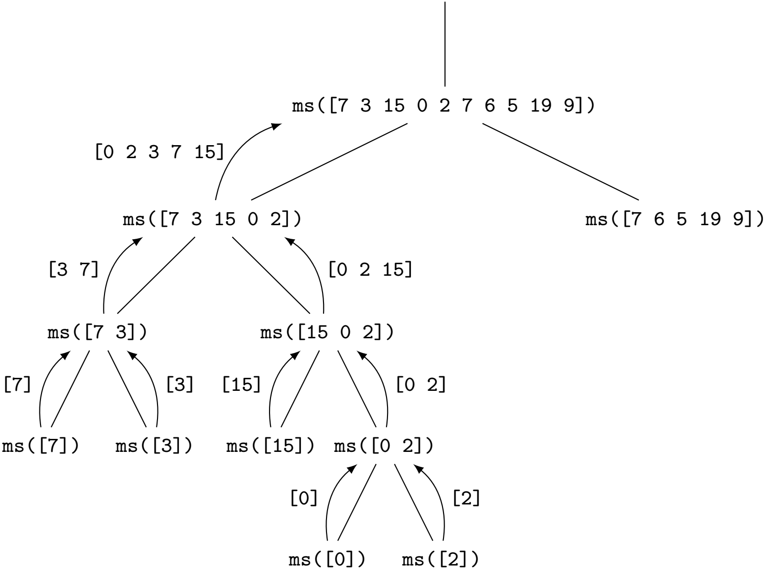
merge sort, anropsträd



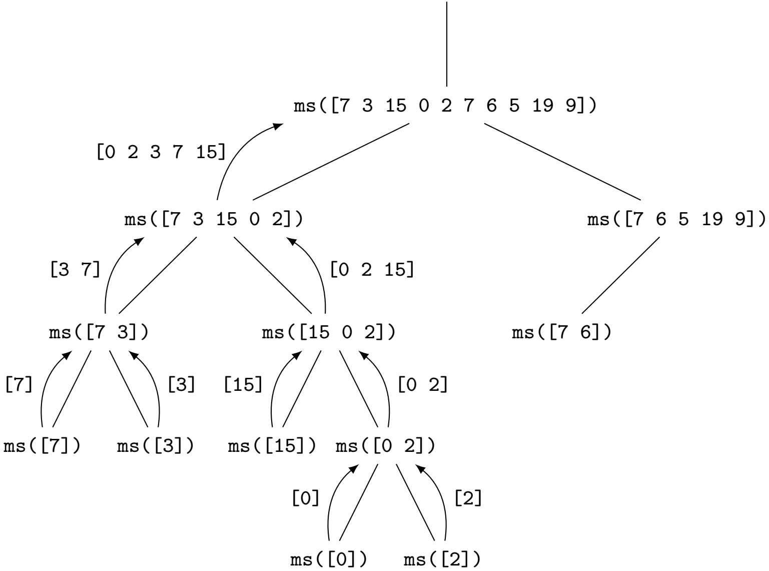
merge sort, anropsträd



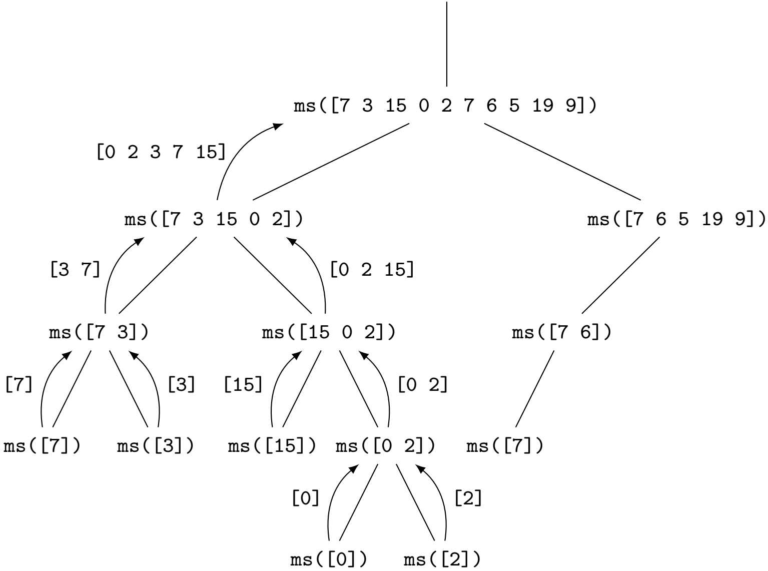
merge sort, anropsträd



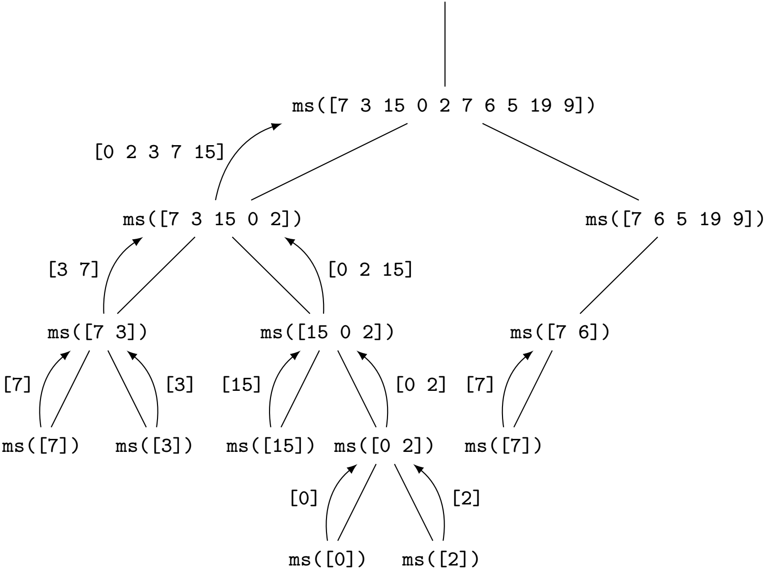
merge sort, anropsträd



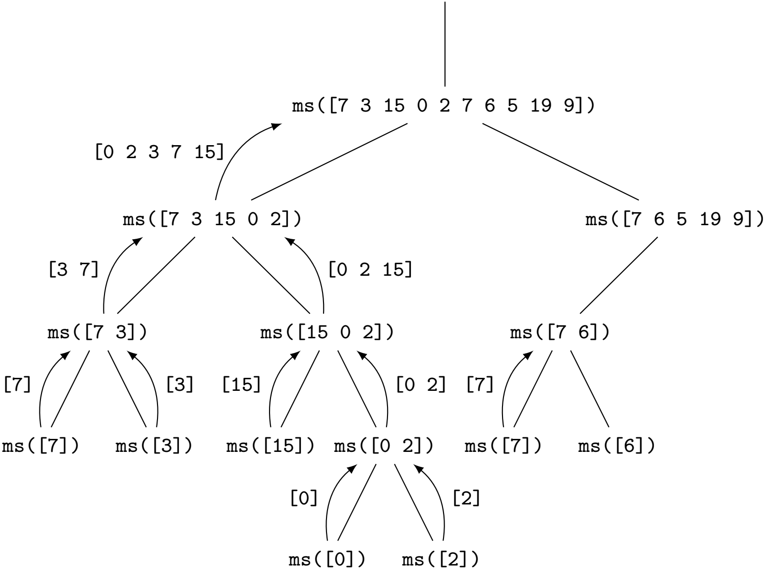
merge sort, anropsträd



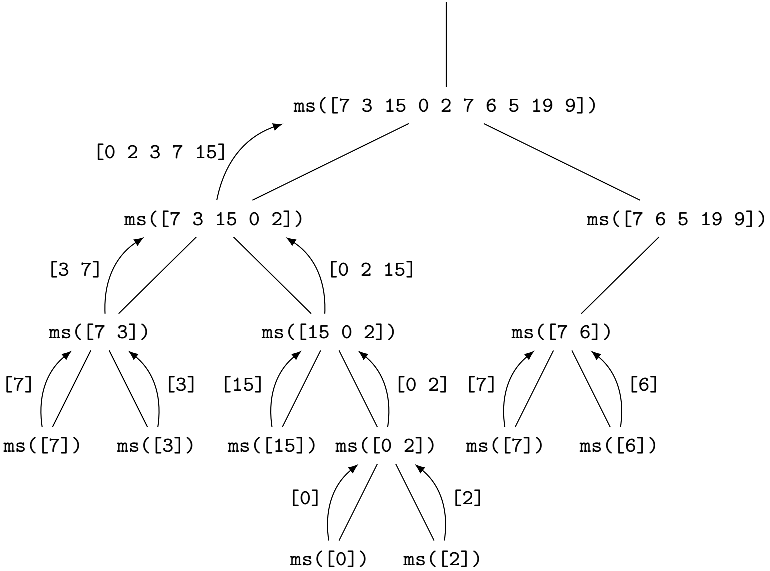
merge sort, anropsträd


ms([7 3 15 0 2 7 6 5 19 9])
[0 2 3 7 15]
ms([7 3 15 0 2])
ms([7 6 5 19 9])
[3 7]
[0 2 15]
ms([7 3])
ms([15 0 2])
ms([7 6])
[7]
[3]
[15]
[0 2]
[7]
ms([7])
ms([3])
ms([15])
ms([0 2])
ms([7])
[0]
[2]
ms([0])
ms([2])

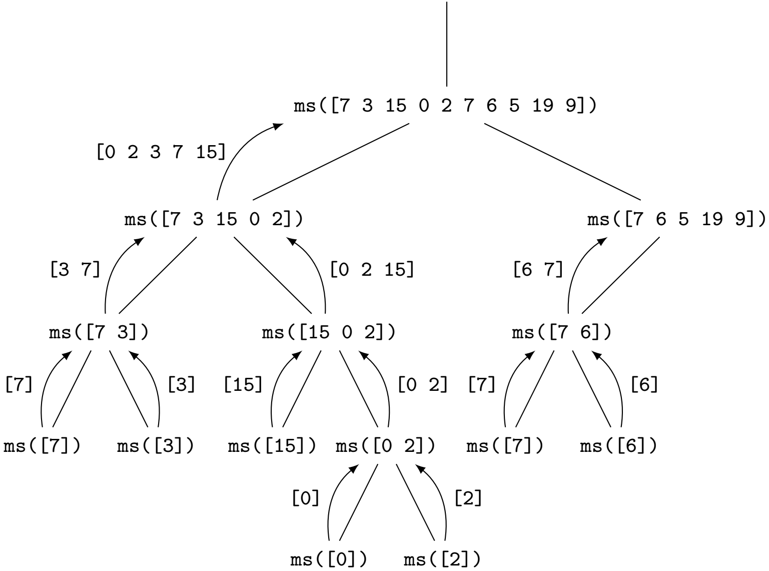
merge sort, anropsträd



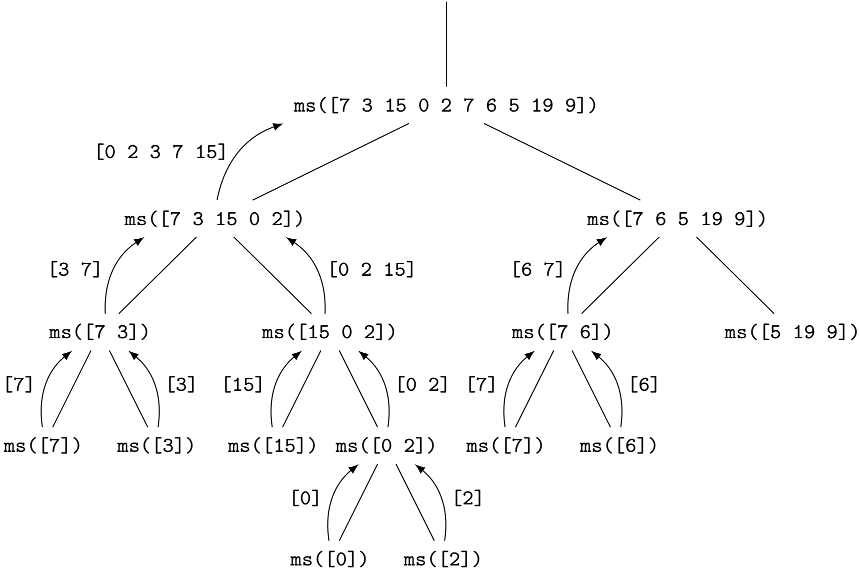
merge sort, anropsträd



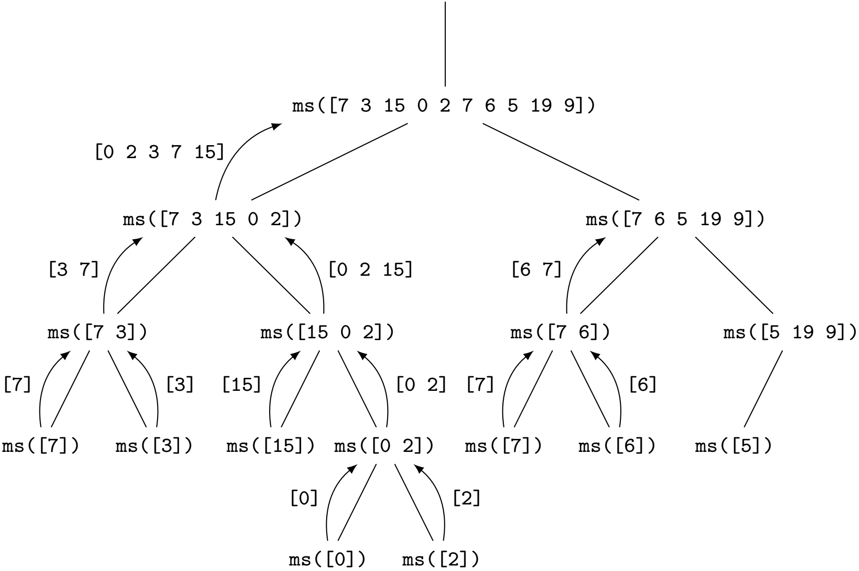
merge sort, anropsträd



merge sort, anropsträd



merge sort, anropsträd



ms([7 3 15 0 2 7 6 5 19 9])

[0 2 3 7 15]

ms([7 3 15 0 2])

ms([7 6 5 19 9])

[3 7]

[0 2 15]

[6 7]

ms([7 3])

ms([15 0 2])

ms([7 6])

ms([5 19 9])

[7]

[3]

[15]

[0 2]

[7]

[6]

ms([7])

ms([3])

ms([15])

ms([0 2])

ms([7])

ms([6])

ms([5])

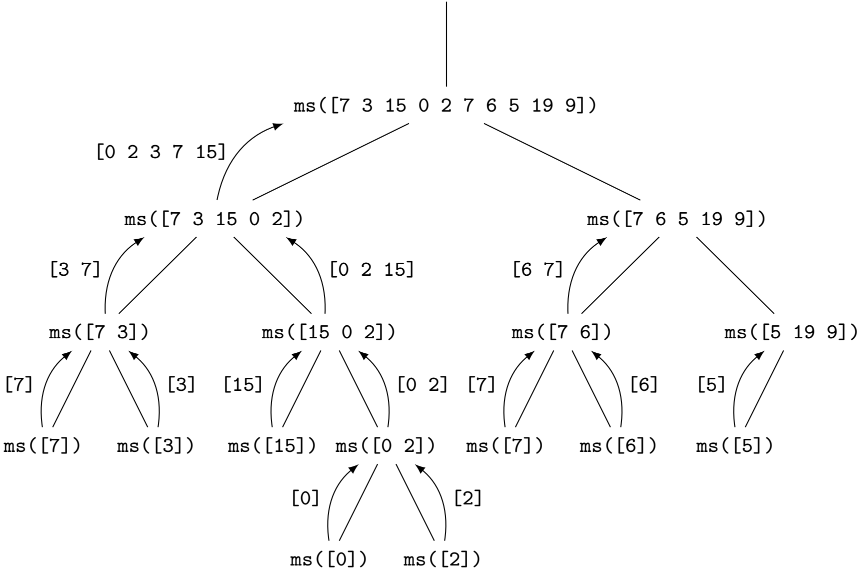
[0]

[2]

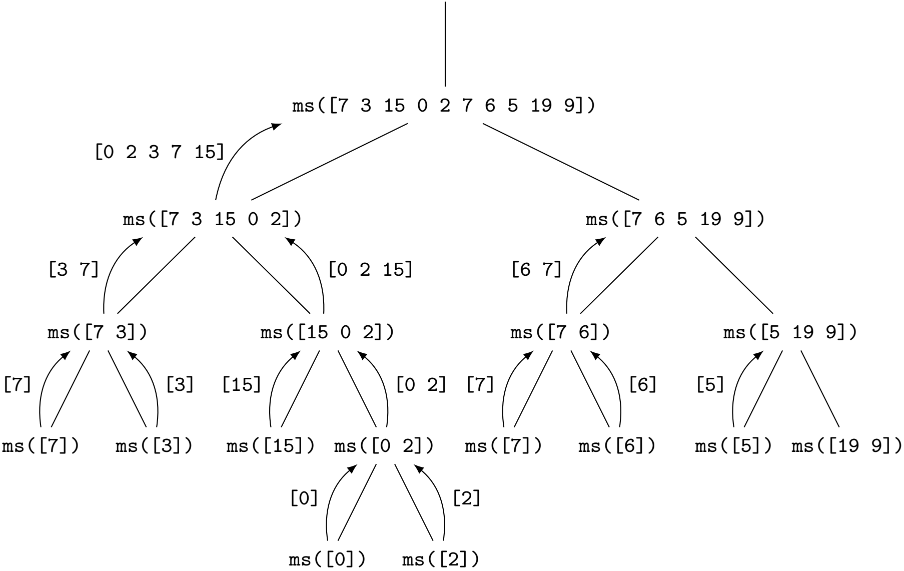
ms([0])

ms([2])

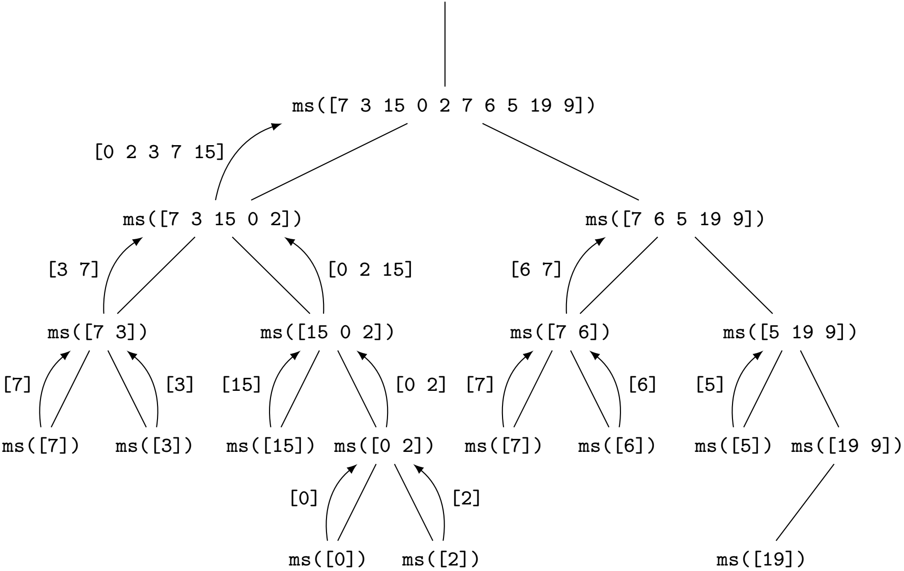
merge sort, anropsträd



merge sort, anropsträd



merge sort, anropsträd



ms([7 3 15 0 2 7 6 5 19 9])

[0 2 3 7 15]

ms([7 3 15 0 2])

ms([7 6 5 19 9])

[3 7]

[0 2 15]

[6 7]

ms([7 3])

ms([15 0 2])

ms([7 6])

ms([5 19 9])

[7]

[3]

[15]

[0 2]

[7]

[6]

[5]

ms([7])

ms([3])

ms([15])

ms([0 2])

ms([7])

ms([6])

ms([5])

ms([19 9])

[0]

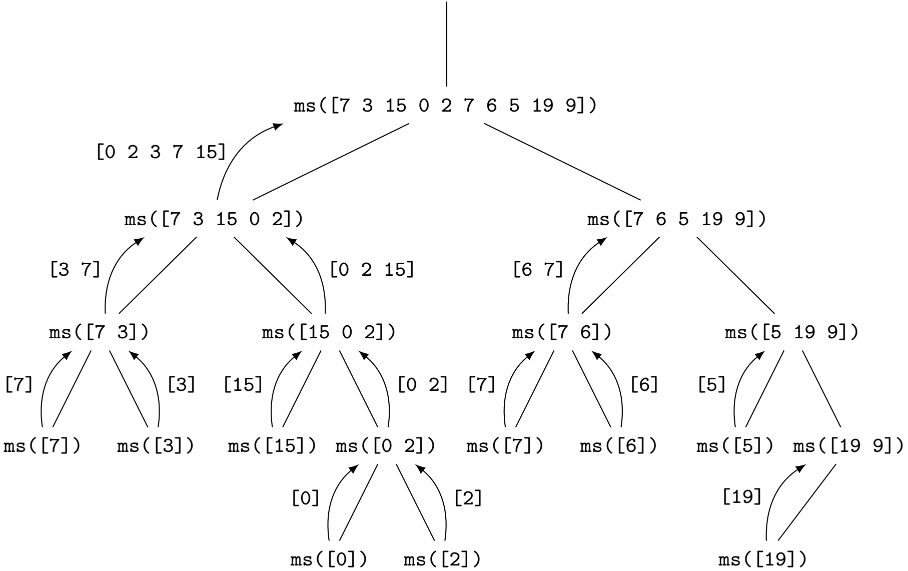
[2]

ms([0])

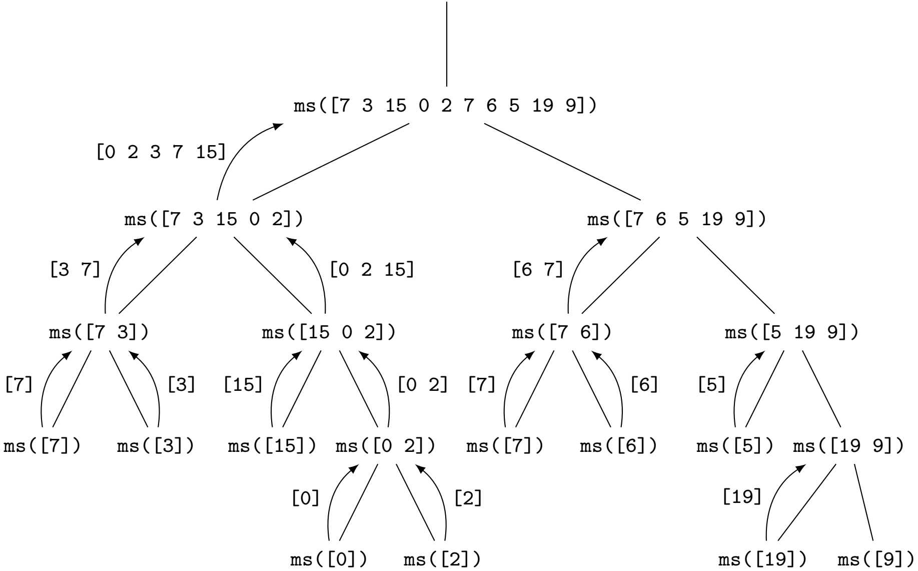
ms([2])

ms([19])

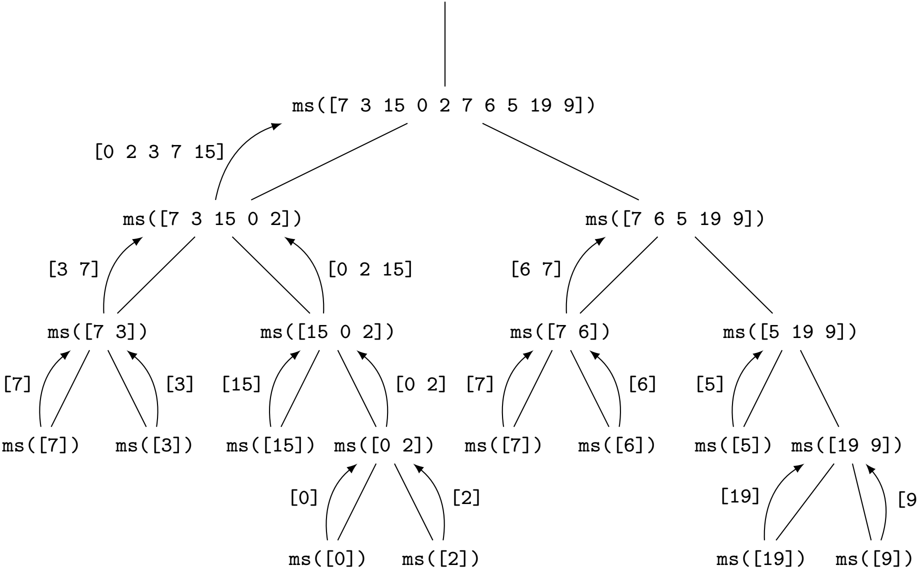
merge sort, anropsträd



merge sort, anropsträd



merge sort, anropsträd



ms([7 3 15 0 2 7 6 5 19 9])

[0 2 3 7 15]

ms([7 3 15 0 2]) ms([7 6 5 19 9])

[3 7] [0 2 15] [6 7]

ms([7 3]) ms([15 0 2]) ms([7 6]) ms([5 19 9])

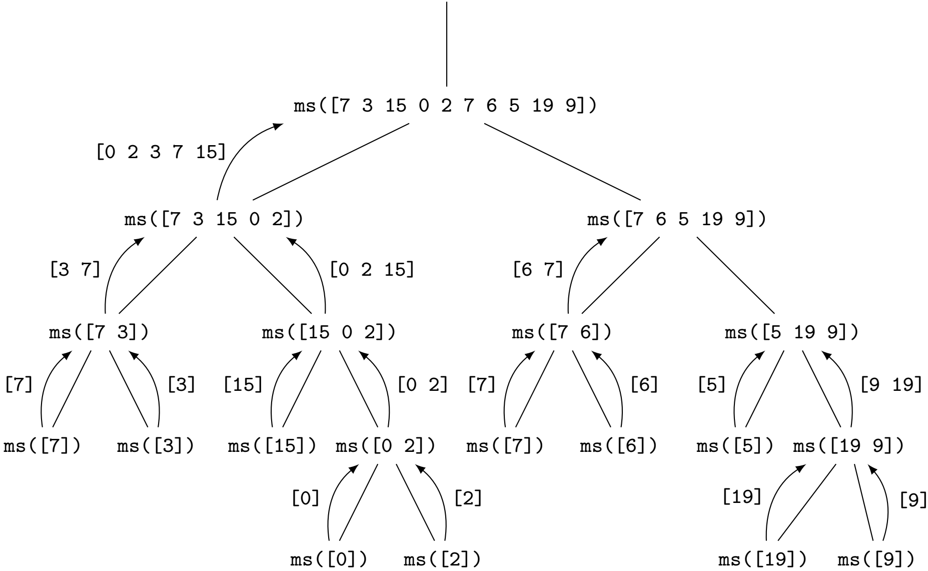
[7] [3] [15] [0 2] [7] [6] [5]

ms([7]) ms([3]) ms([15]) ms([0 2]) ms([7]) ms([6]) ms([5]) ms([19 9])

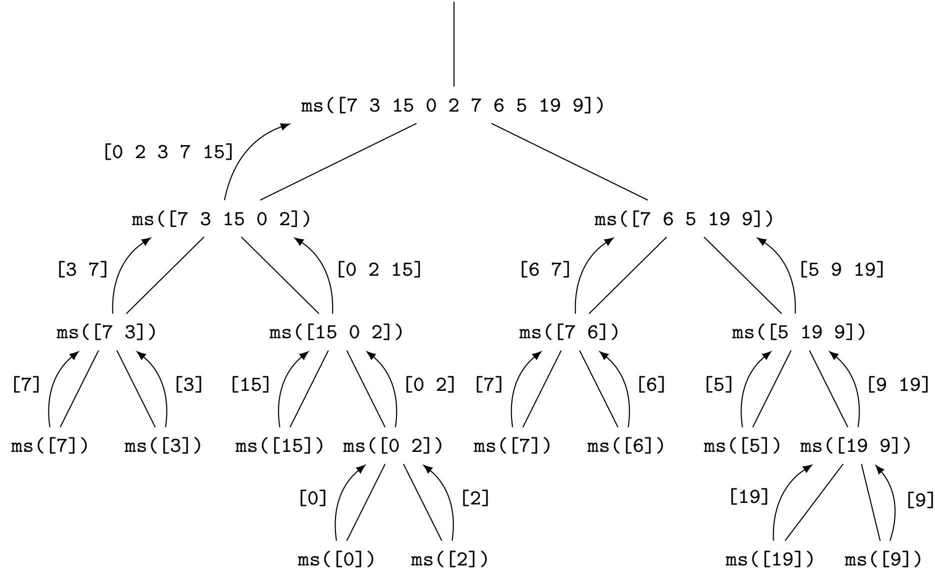
[0] [2] [19] [9]

ms([0]) ms([2]) ms([19]) ms([9])

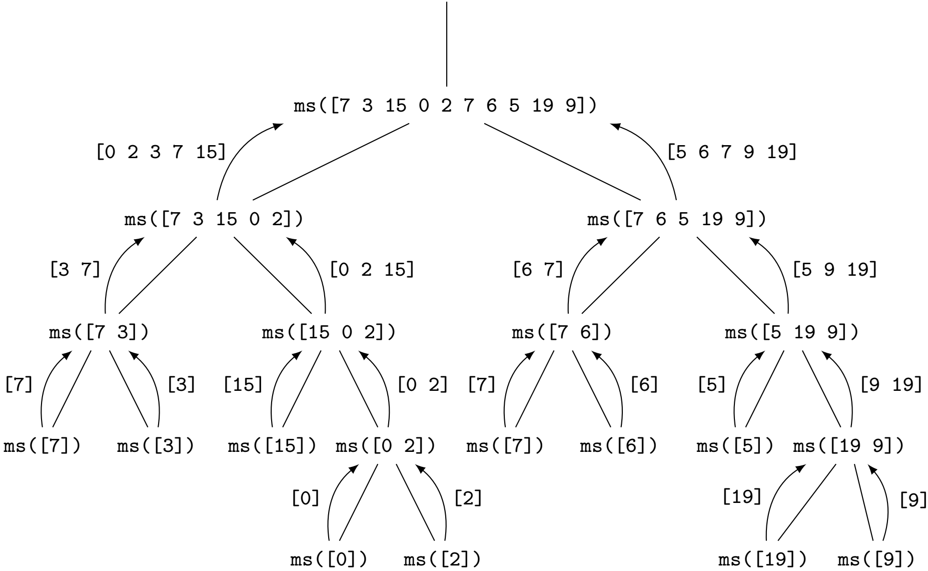
merge sort, anropsträd



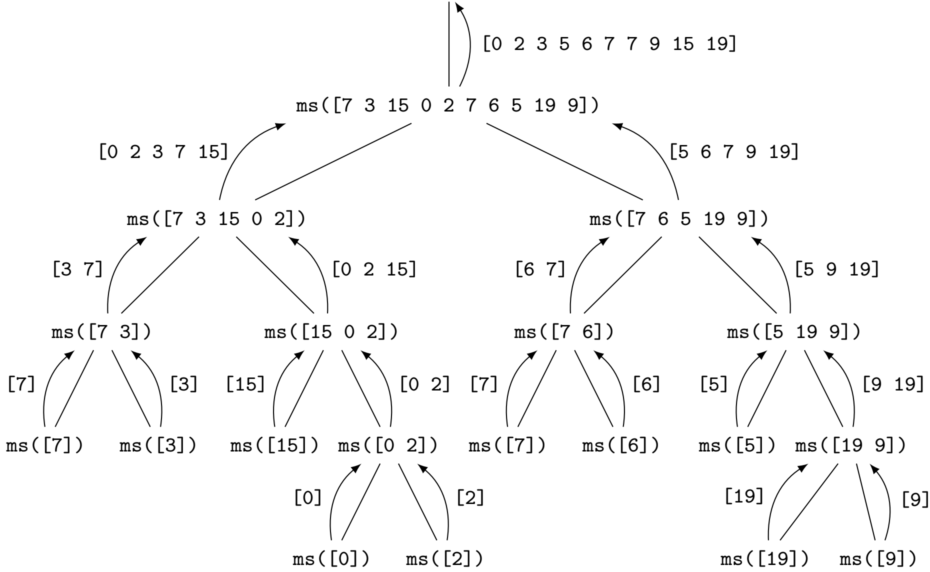
merge sort, anropsträd



merge sort, anropsträd



merge sort, anropsträd



Sortering, summering

- Varför olika algoritmer?
  - Olika effektivitet
  - Olika problemlösningsstrategier
    - *Insertion Sort* — instickssortering
    - *Bubbel Sort* — utbytessortering
    - *Merge Sort* — samsortering
- Det finns fler algoritmer än dessa
  - Mer på *Datastrukturer och Algoritmer*-kursen
  - Sök på "hungarian folk dance sorting" på youtube
    - https://www.youtube.com/watch?v=dENca26N6V4
    - https://www.youtube.com/watch?v=EdIKIf9mHk0&list=PLOmdoKois7_FK-ySGwHBkltzB11snW7KQ