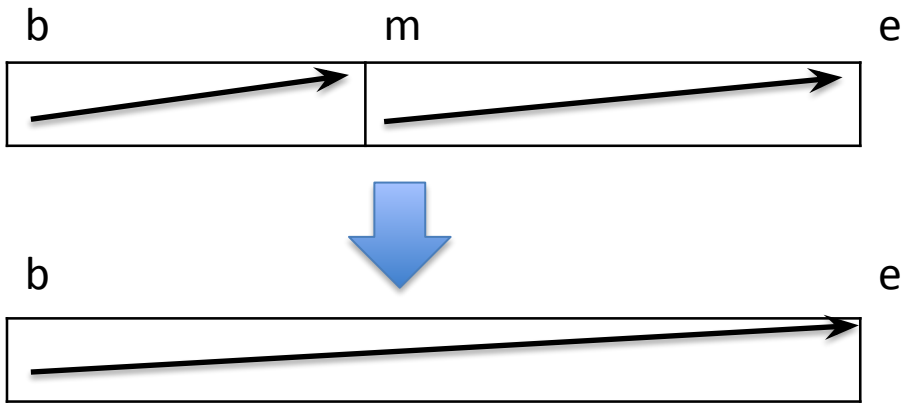
Tracing through sort

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

void merge(int a[], int b, int m, int e)
{
```




```
}
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

int main()
{  int arr[5] = { 40, 30, 20, 50, 10 };
   sort(arr, 0, 5);
  D...
}
```

environment of
main:

	[0]	[1]	[2]	[3]	[4]
arr:	40	30	20	50	10

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

int main()
{  int arr[5] = { 40, 30, 20, 50, 10 };
   sort(arr, 0, 5);
  D...
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   40   30   20   50   10

sort1:     a:      b: 0    e: 5          return to D
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
   Asort(a, mid, e);    // sort right half
   Bmerge(a, b, mid, e); // merge two halves
   C}
}

main:        [0] [1] [2] [3] [4]
       arr:  40  30  20  50  10

sort1:       b: 0   e: 5   mid: 2        return to D
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

main:         [0] [1] [2] [3] [4]
       arr:   40  30  20  50  10

sort1:        b: 0   e: 5   mid: 2        return to D

sort2:        b: 0   e: 2               return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
   Asort(a, mid, e);    // sort right half
   Bmerge(a, b, mid, e); // merge two halves
   C}
}
```

```
main:          [0] [1] [2] [3] [4]
         arr:   40  30  20  50  10

sort1:         b: 0   e: 5   mid: 2        return to D

sort2:         b: 0   e: 2   mid: 1        return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   40  30  20  50  10

sort1:       b: 0   e: 5   mid: 2        return to D

sort2:       b: 0   e: 2   mid: 1        return to A

sort3:       b: 0   e: 1                 return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
A     sort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:          [0] [1] [2] [3] [4]
        arr:    40  30  20  50  10

sort1:         b: 0   e: 5   mid: 2        return to D

sort2:         b: 0   e: 2   mid: 1        return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   40   30   20   50   10
```

sort$_1$: b: 0 e: 5 mid: 2 return to D

sort$_2$: b: 0 e: 2 mid: 1 return to A

sort$_4$: b: 1 e: 2 return to B

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
□B    merge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   40  30  20  50  10

sort1:       b: 0   e: 5   mid: 2        return to D

sort2:       b: 0   e: 2   mid: 1        return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   40  30  20  50  10
              ~~  ~~
sort1:       b: 0   e: 5   mid: 2       return to D

sort2:       b: 0   e: 2   mid: 1       return to A

merge1:      b: 0   m: 1   e: 2      return to C
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
C }
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  50  10

sort1:       b: 0   e: 5   mid: 2       return to D

sort2:       b: 0   e: 2   mid: 1       return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
A     sort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

main:        [0] [1] [2] [3] [4]
       arr:  30  40  20  50  10

sort1:      b: 0   e: 5   mid: 2        return to D
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{ if (e – b >= 2)
    {  int mid = (b + e) / 2;
       sort(a, b, mid);  // sort left half
   Asort(a, mid, e);     // sort right half
   Bmerge(a, b, mid, e); // merge two halves
   C}
}

main:          [0] [1] [2] [3] [4]
         arr:   30  40  20  50  10

sort1:         b: 0   e: 5   mid: 2        return to D

sort5:         b: 2   e: 5               return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
   Asort(a, mid, e);    // sort right half
   Bmerge(a, b, mid, e); // merge two halves
   C}
}

main:          [0] [1] [2] [3] [4]
        arr:    30  40  20  50  10

sort1:         b: 0   e: 5   mid: 2        return to D

sort5:         b: 2   e: 5   mid: 3     return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

main:         [0] [1] [2] [3] [4]
       arr:   30  40  20  50  10

sort1:        b: 0   e: 5   mid: 2        return to D

sort5:        b: 2   e: 5   mid: 3     return to B

sort6:        b: 2   e: 3              return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
A     sort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  50  10

sort1:       b: 0   e: 5   mid: 2        return to D

sort5:       b: 2   e: 5   mid: 3    return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  50  10

sort1:       b: 0   e: 5   mid: 2        return to D

sort5:       b: 2   e: 5   mid: 3     return to B

sort7:       b: 3   e: 5              return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
   Asort(a, mid, e);    // sort right half
   Bmerge(a, b, mid, e); // merge two halves
   C}
}
```

```
main:          [0] [1] [2] [3] [4]
         arr:   30  40  20  50  10

sort1:         b: 0   e: 5   mid: 2        return to D

sort5:         b: 2   e: 5   mid: 3     return to B

sort7:         b: 3   e: 5   mid: 4     return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  50  10

sort1:       b: 0   e: 5   mid: 2       return to D

sort5:       b: 2   e: 5   mid: 3    return to B

sort7:       b: 3   e: 5   mid: 4    return to B

sort8:       b: 3   e: 4             return to A
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
A     sort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:          [0] [1] [2] [3] [4]
         arr:   30  40  20  50  10

sort1:         b: 0   e: 5   mid: 2        return to D

sort5:         b: 2   e: 5   mid: 3     return to B

sort7:         b: 3   e: 5   mid: 4     return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  50  10

sort1:       b: 0   e: 5   mid: 2       return to D

sort5:       b: 2   e: 5   mid: 3    return to B

sort7:       b: 3   e: 5   mid: 4    return to B

sort9:       b: 4   e: 5             return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
B    merge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:          [0] [1] [2] [3] [4]
        arr:    30  40  20  50  10

sort1:         b: 0   e: 5   mid: 2        return to D

sort5:         b: 2   e: 5   mid: 3     return to B

sort7:         b: 3   e: 5   mid: 4     return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  50  10
                              ~~  ~~
sort1:       b: 0   e: 5   mid: 2       return to D

sort5:       b: 2   e: 5   mid: 3    return to B

sort7:       b: 3   e: 5   mid: 4    return to B

merge2:      b: 3   m: 4   e: 5      return to C
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
C }
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  10  50

sort1:       b: 0   e: 5   mid: 2       return to D

sort5:       b: 2   e: 5   mid: 3    return to B

sort7:       b: 3   e: 5   mid: 4    return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
B    merge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  20  10  50

sort1:       b: 0   e: 5   mid: 2       return to D

sort5:       b: 2   e: 5   mid: 3    return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
B    merge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:          [0] [1] [2] [3] [4]
        arr:    30  40  20  10  50
                        ~~  ~~~~~~
sort1:        b: 0   e: 5   mid: 2        return to D

sort5:        b: 2   e: 5   mid: 3     return to B

merge3:       b: 2   m: 3   e: 5       return to C
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
C }
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  10  20  50

sort1:       b: 0   e: 5   mid: 2        return to D

sort5:       b: 2   e: 5   mid: 3    return to B
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
B    merge(a, b, mid, e); // merge two halves
  C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   30  40  10  20  50

sort1:       b: 0   e: 5   mid: 2        return to D
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

main:         [0] [1] [2] [3] [4]
       arr:    30  40  10  20  50
               ~~~~~~  ~~~~~~~~~~
sort1:       b: 0   e: 5   mid: 2        return to D

merge4:      b: 0   m: 2   e: 5       return to C
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
C}
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   10  20  30  40  50

sort1:       b: 0   e: 5   mid: 2        return to D
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
C}
}

int main()
{  int arr[5] = { 40, 30, 20, 50, 10 };
   sort(arr, 0, 5);
  D...
}
```

```
main:        [0] [1] [2] [3] [4]
       arr:   10  20  30  40  50

sort1:       b: 0   e: 5   mid: 2        return to D
```

```
void sort(int a[], int b, int e) // sort a[b] to a[e-1]
{  if (e – b >= 2)
   {  int mid = (b + e) / 2;
      sort(a, b, mid);  // sort left half
  Asort(a, mid, e);     // sort right half
  Bmerge(a, b, mid, e); // merge two halves
  C}
}

int main()
{  int arr[5] = { 40, 30, 20, 50, 10 };
   sort(arr, 0, 5);
D ...
}

main:        [0] [1] [2] [3] [4]
       arr:  10  20  30  40  50
```