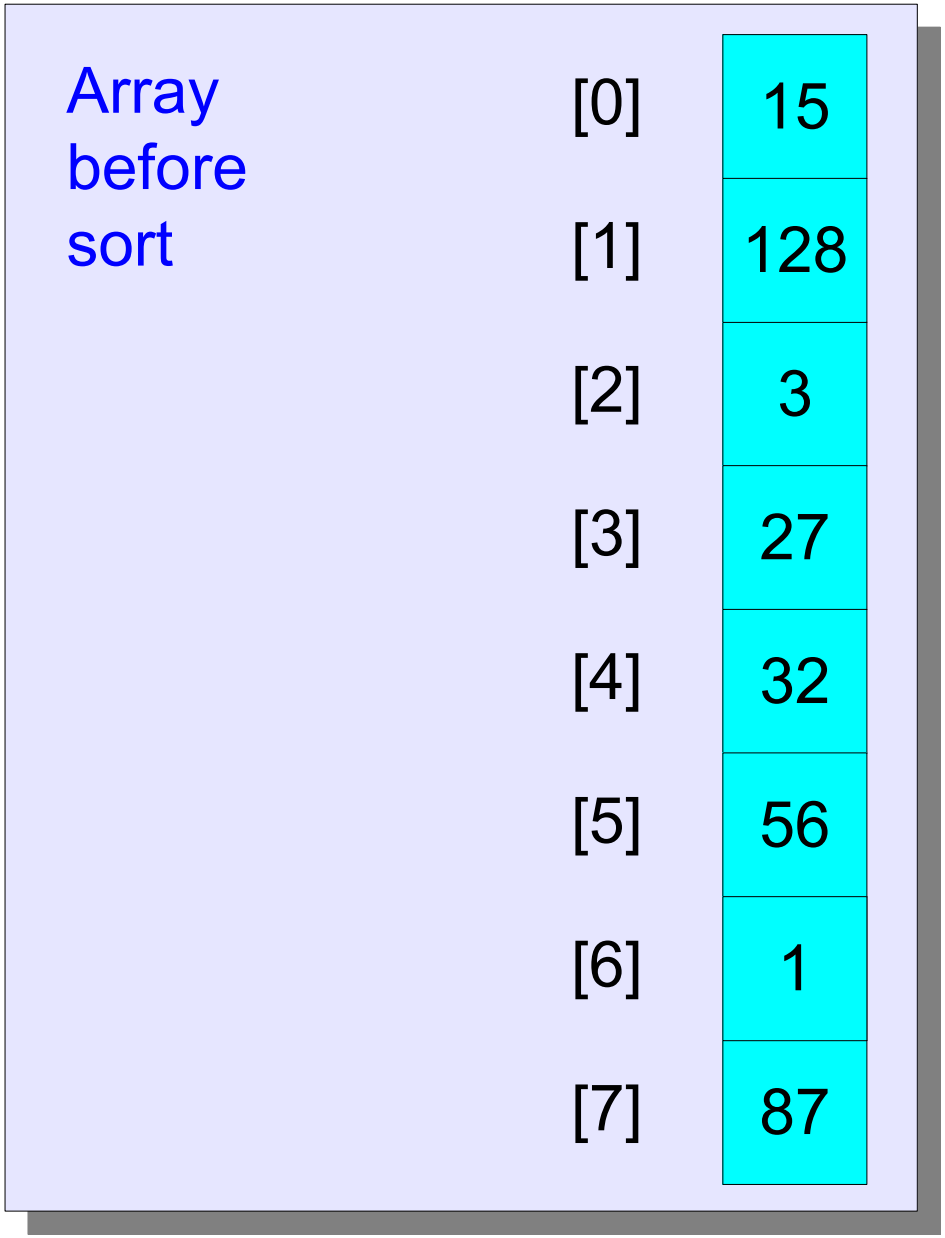
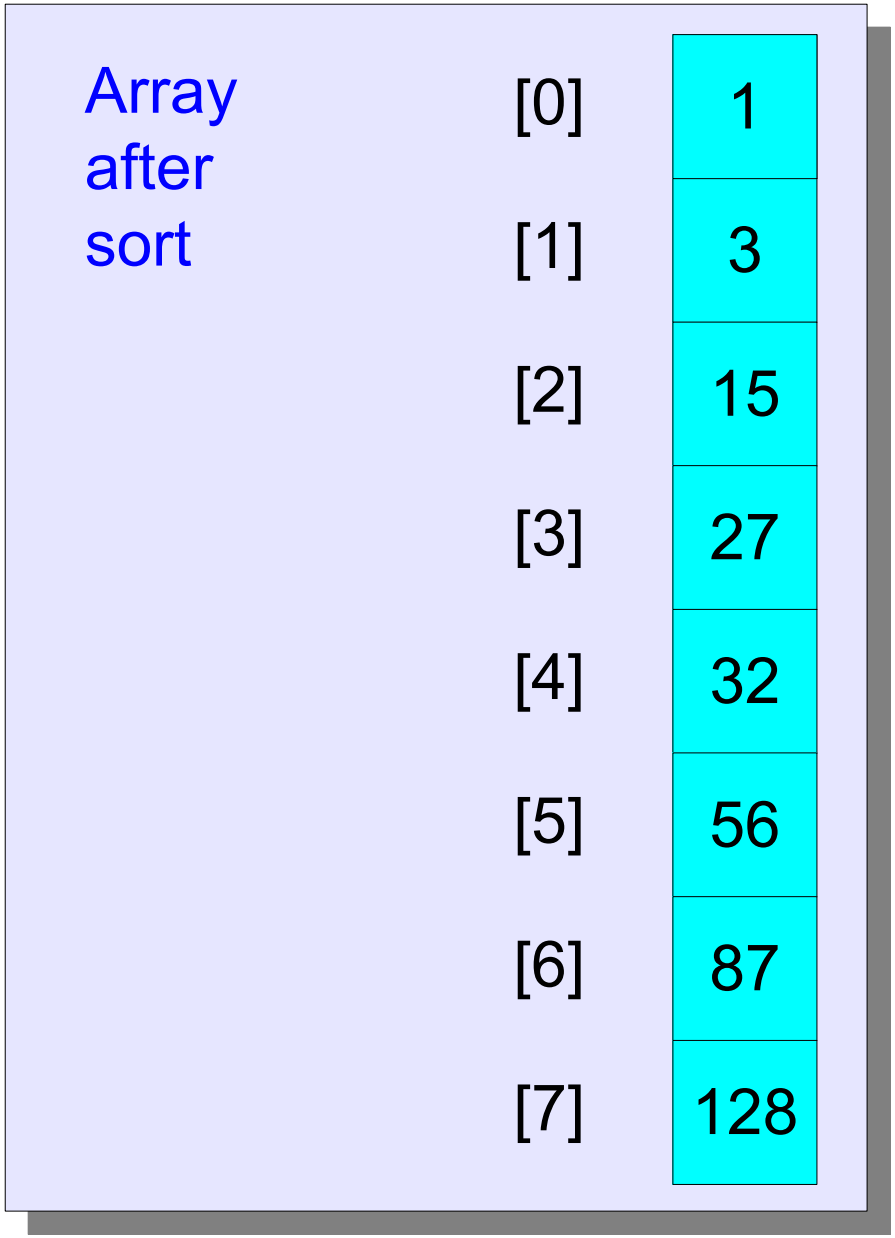
Sorting

- Arranging records according to a specified sequence, such as alphabetically or numerically, from lowest to highest.

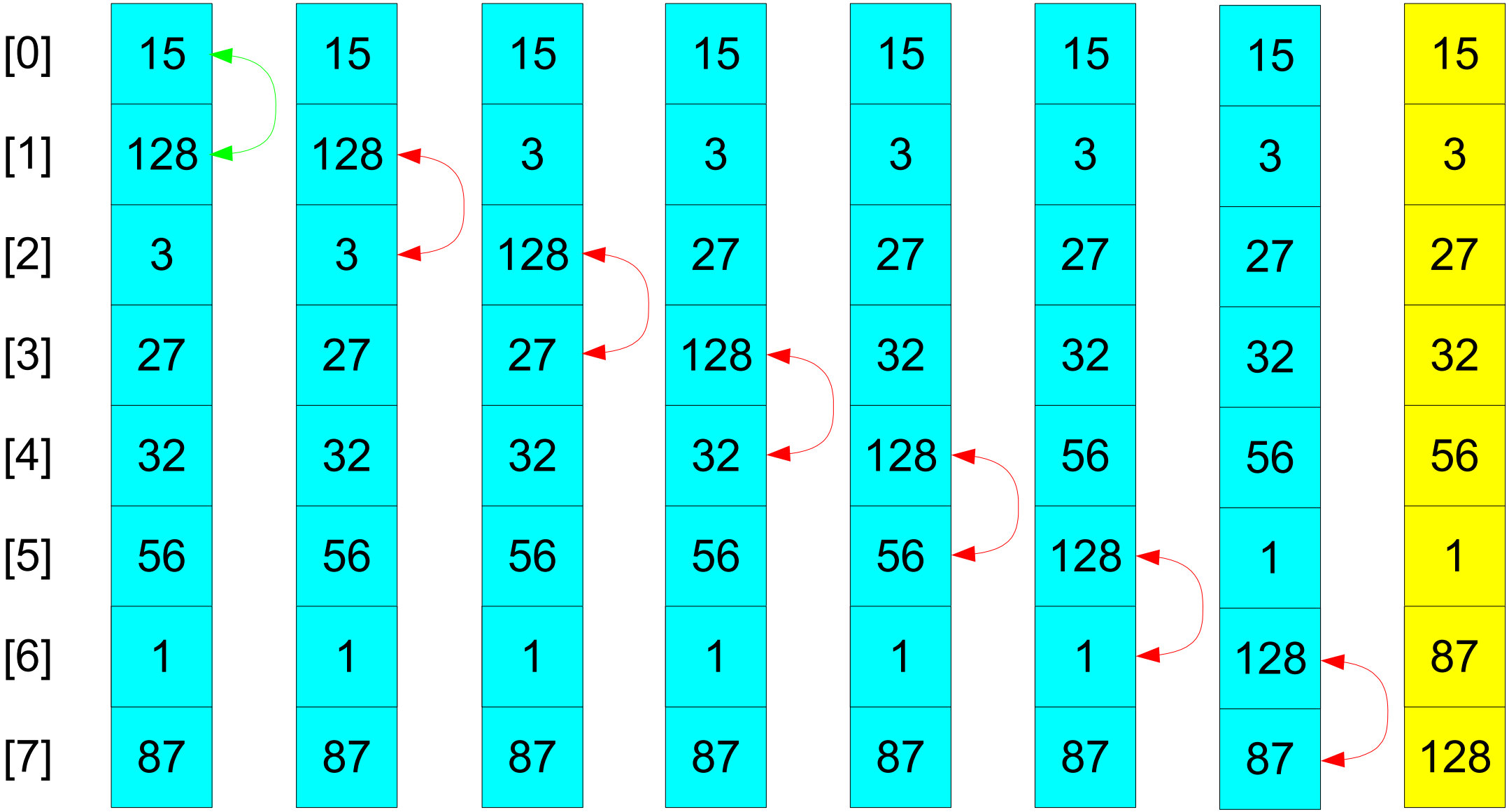




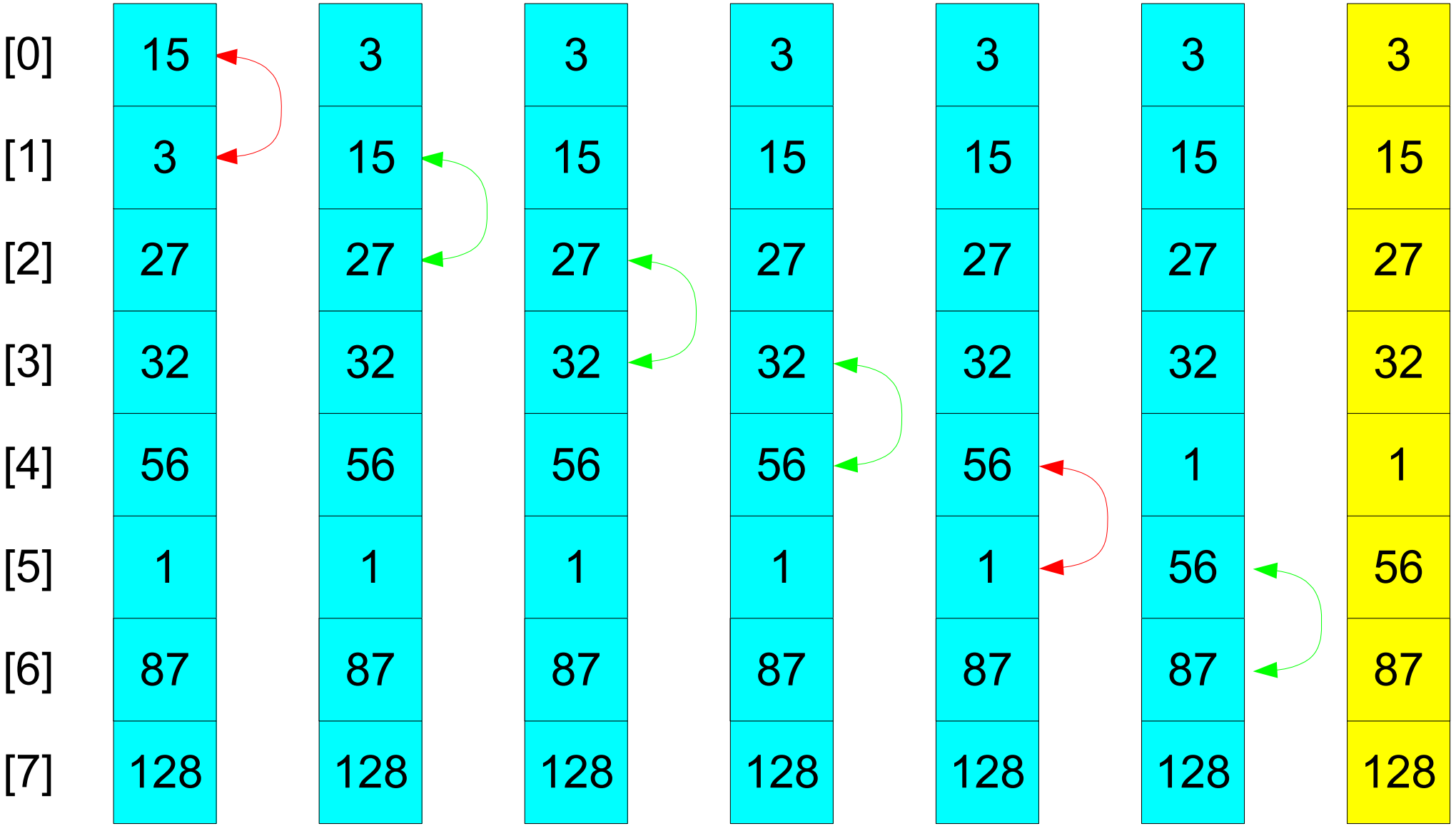
Bubblesort

- Bubblesort is one of the simplest (and one of the least effective) sorting algorithms.
- Bubble sort compares two values next to each other and exchanges them if necessary to put them in the correct order. There are many variations on bubble sort, but they all examine adjacent pairs.
- The larger values gradually bubble their way up to the top like air bubbles in water.
- Several passes throughout an array are necessary to complete the sort.

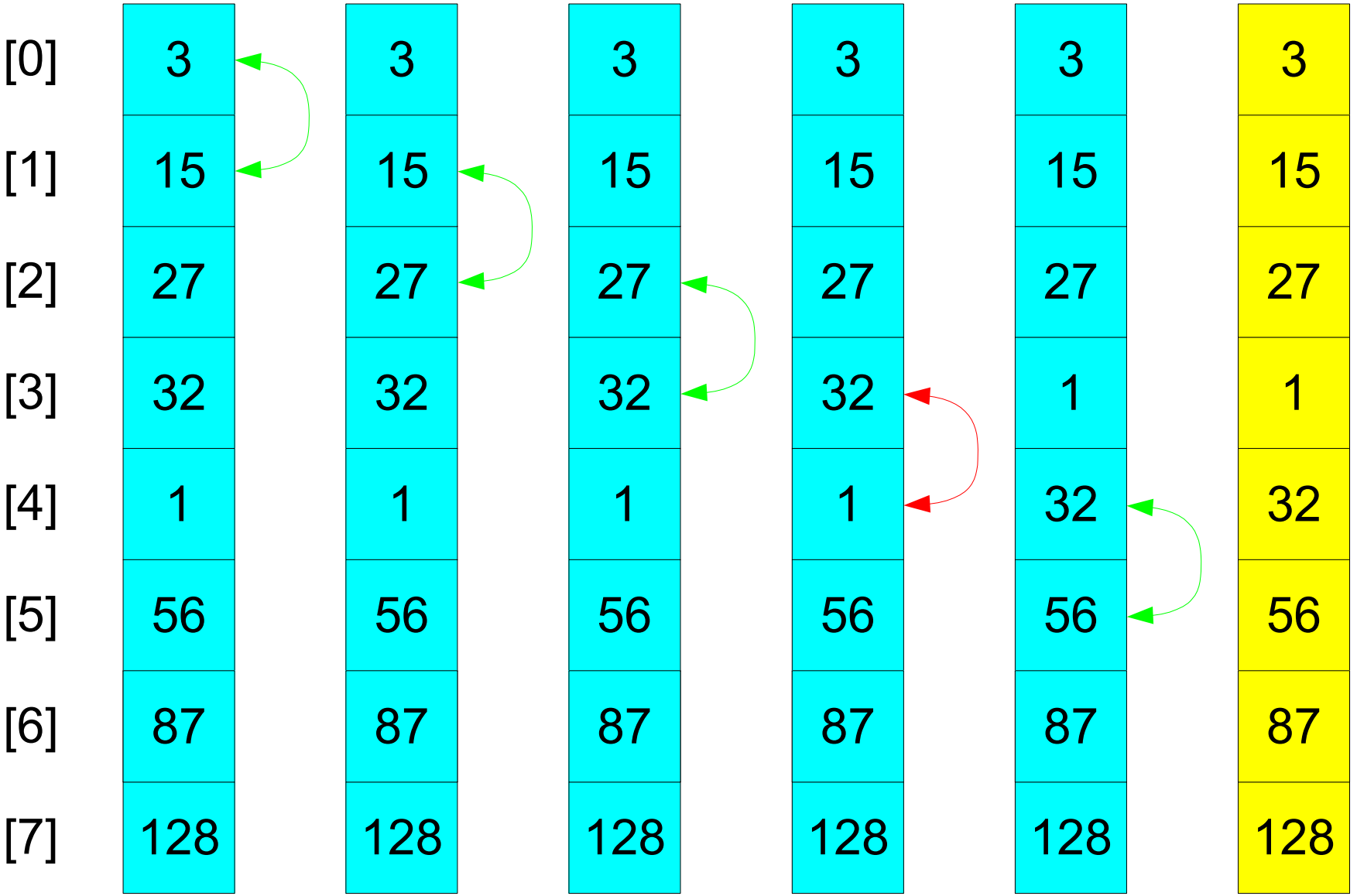
Bubblesort (pass 1)



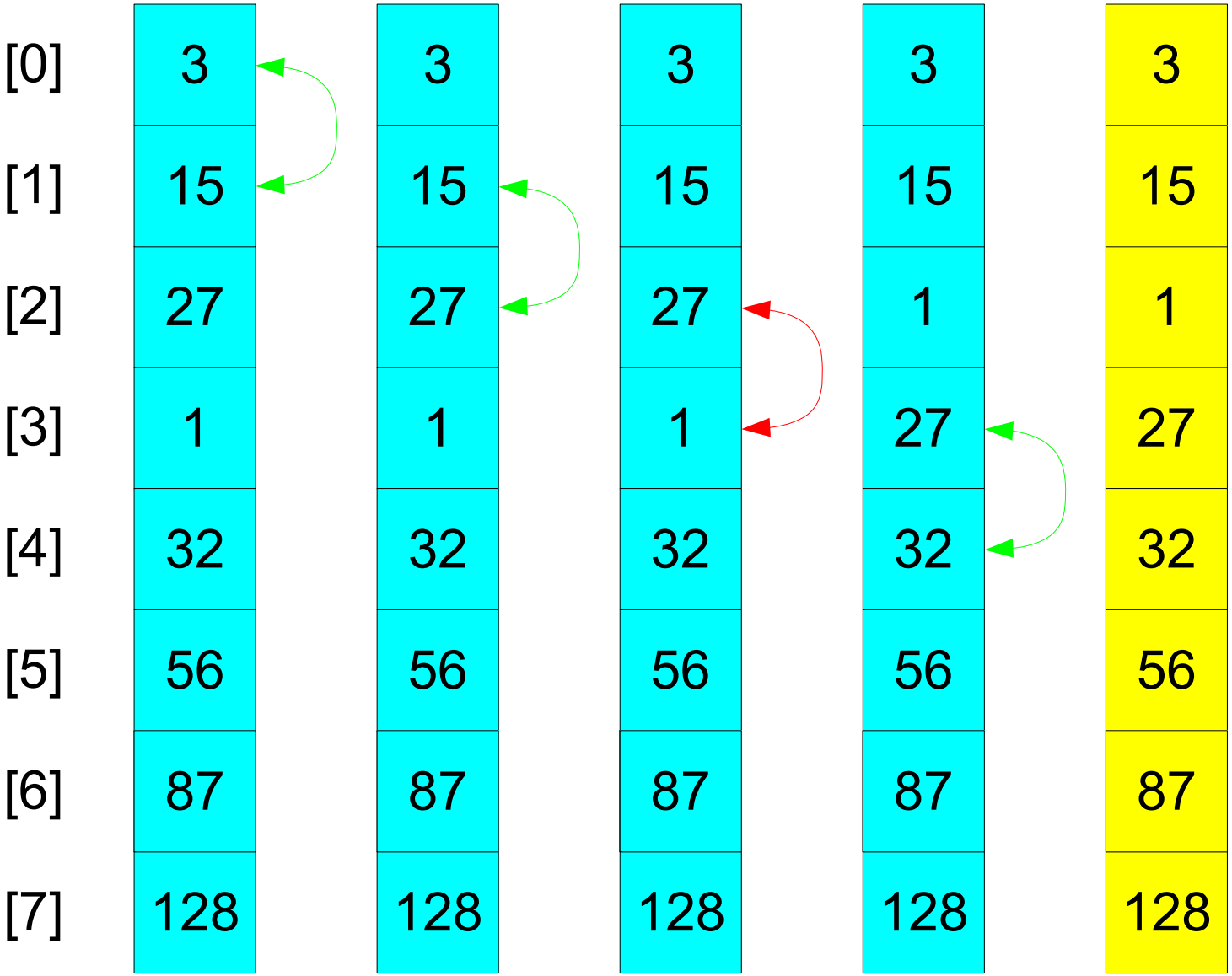
Bubblesort (pass 2)



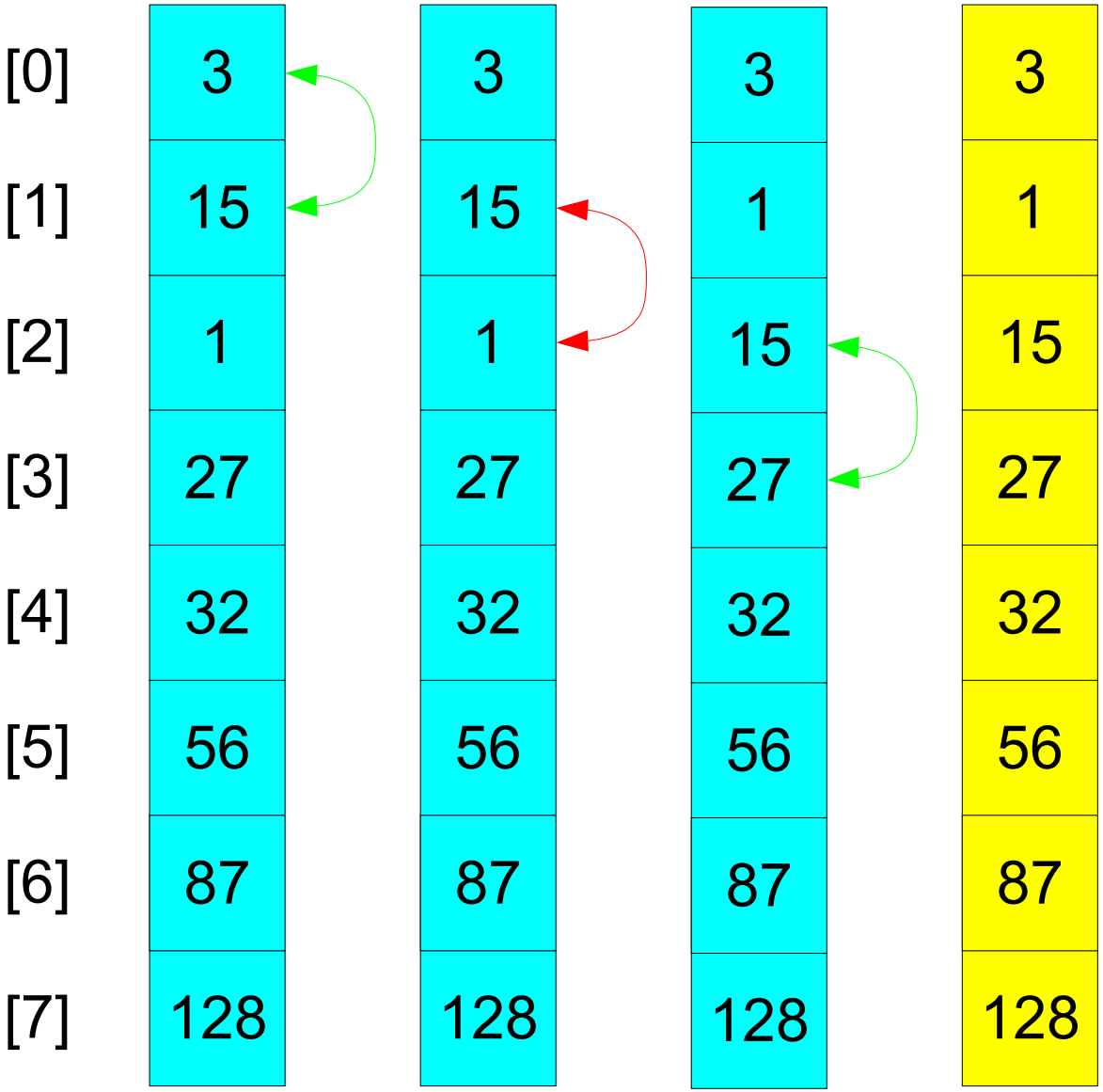
Bubblesort (pass 3)



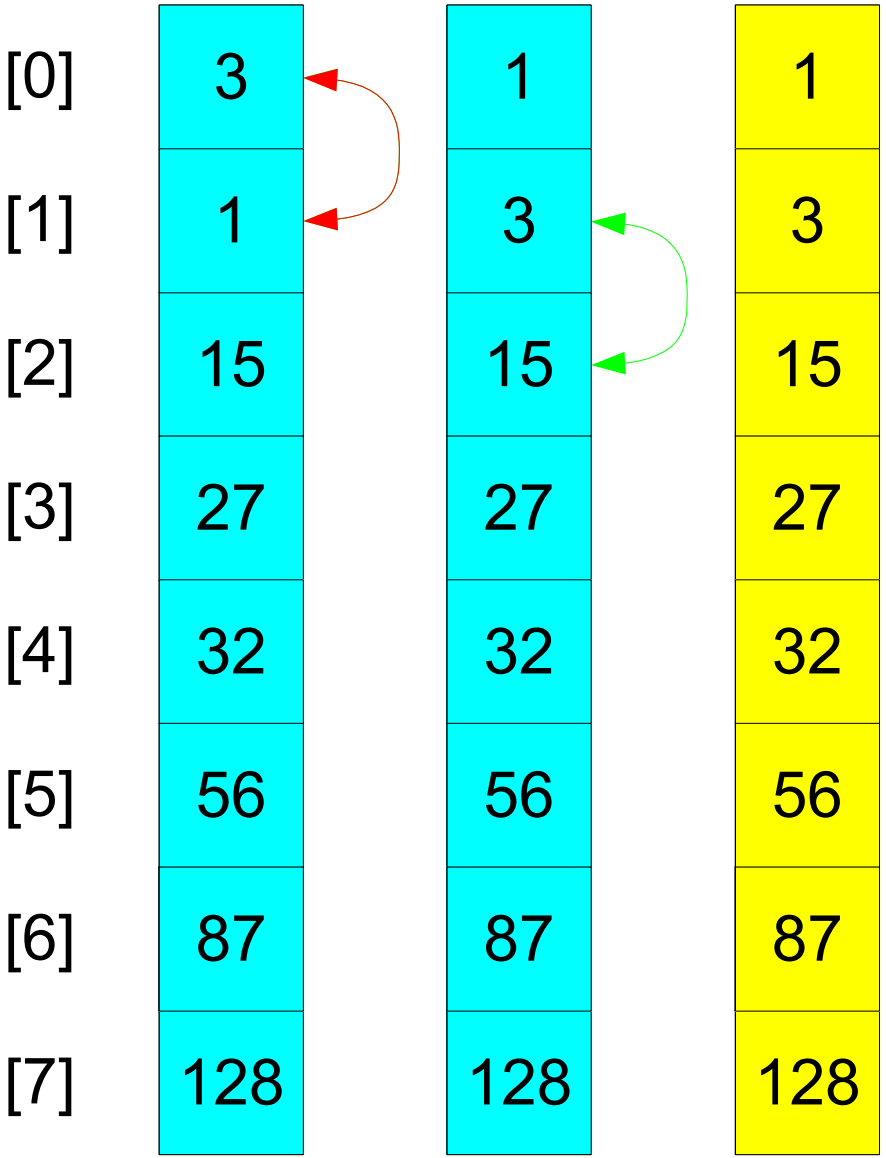
Bubblesort (pass 4)



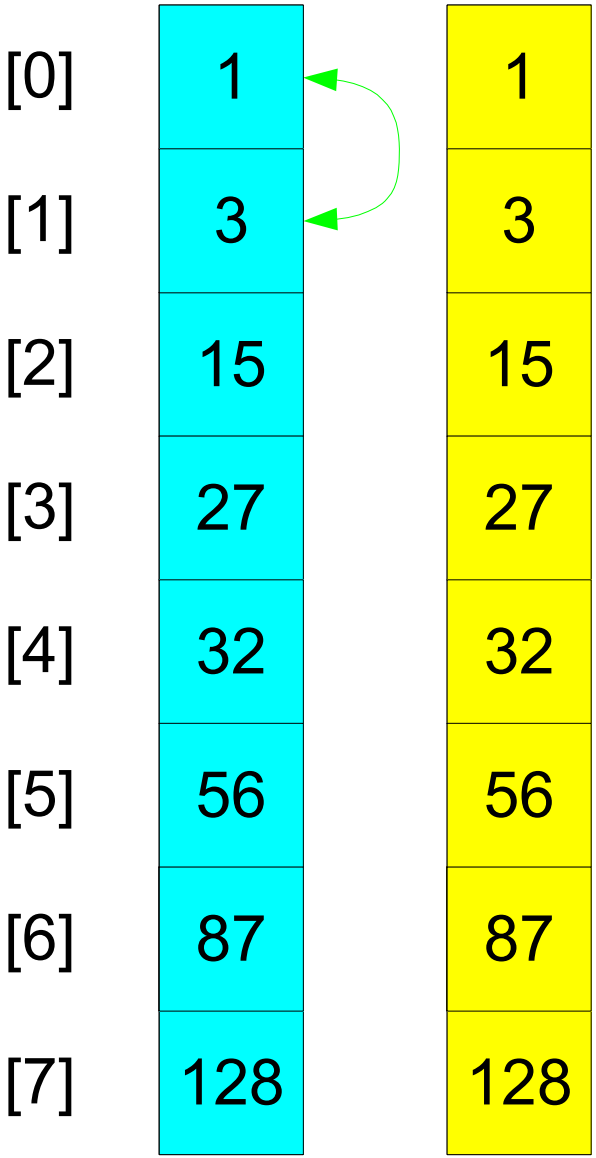
Bubblesort (pass 5)



Bubblesort (pass 6)



Bubblesort (pass 7)



Bubblesort - C Code

```
#include <stdio.h>

void display (int *a, unsigned int n) {
  unsigned int i;
  printf ("Printing array\n");
  for (i = 0; i < n; i++)
    printf ("%d\n", a[i]);
}

void swap (int *a, int *b) {
  int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}

void bubblesort (int *a, unsigned int n) {
  unsigned int i, j;
  for (i = n - 1; i > 0; i--)
    for (j = 0; j < i; j++)
      if (a[j] > a[j + 1])
        swap (a + j, a + j + 1);
}

int main () {
  int numbers[] = { 15, 128, 3, 27, 32, 56, 1, 87 };
  display (numbers, sizeof (numbers) / sizeof (*numbers));
  bubblesort (numbers, sizeof (numbers) / sizeof (*numbers));
  display (numbers, sizeof (numbers) / sizeof (*numbers));
  return 0;
}
```

Bubblesort - C Code for Accounts

- To sort customer accounts, we have to change the swap function and the comparison

```
typedef struct
{
  char name[21];
  float balance;
  int accNo;
}
CustAccount;

void
swap (CustAccount * a, CustAccount * b)
{
  CustAccount tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}

void
bubblesort (CustAccount * a, unsigned int n)
{
  unsigned int i, j;
  for (i = n - 1; i > 0; i--)
    for (j = 0; j < i; j++)
      if (strcmp (a[j].name, a[j + 1].name) > 0)
        swap (a + j, a + j + 1);
}
```

Bubblesort - C Code for Accounts

- The pointer to comparison function can be a parameter to the sort function to allow different sort criteria

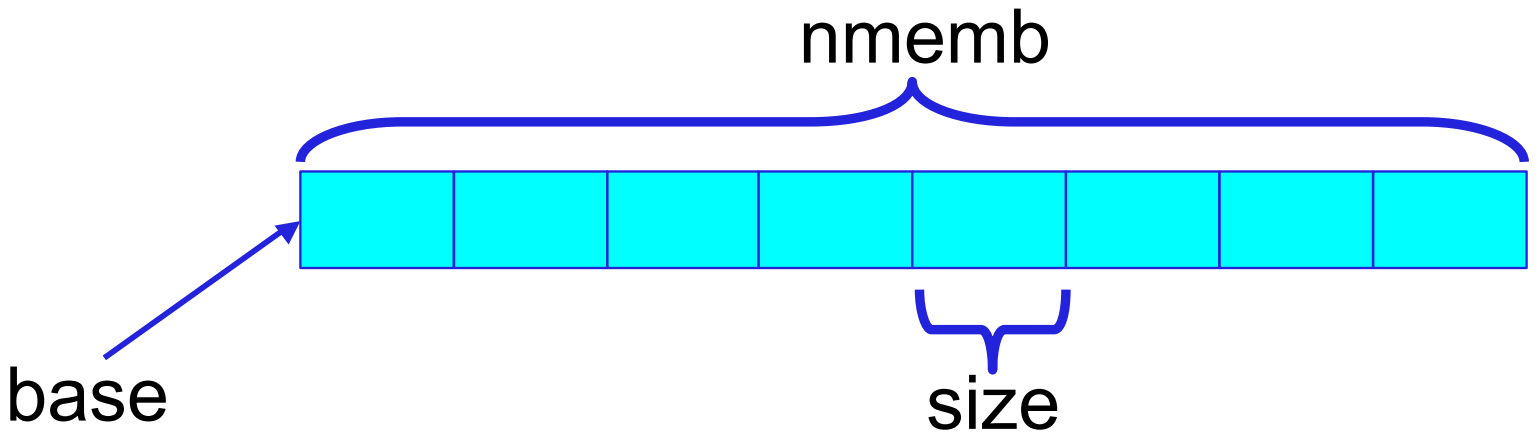
```
void swap (CustAccount * a, CustAccount * b) {
  CustAccount tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}

void bubblesort (CustAccount * a, unsigned int n,
            int (*comp) (const CustAccount *, const CustAccount *)) {
  unsigned int i, j;
  for (i = n - 1; i > 0; i--)
    for (j = 0; j < i; j++)
      if (comp (a + j, a + j + 1) > 0)
        swap (a + j, a + j + 1);
}

int compAccNo (const CustAccount * p1, const CustAccount * p2) {
  return p1->accNo - p2->accNo;
}

int compName (const CustAccount * p1, const CustAccount * p2) {
  return strcmp (p1->name, p2->name);
}
```

```
CustAccount accounts[] = { ... };
bubblesort (accounts, sizeof (accounts) / sizeof (*accounts), compName);
bubblesort (accounts, sizeof (accounts) / sizeof (*accounts), compAccNo);
```

Bubblesort - General Case

- To perform sort in general case, **`bubblesort`** needs two pieces of information:
  - The size of individual array elements
  - The comparison criterion
- We also have to pass the size to **`swap`**

```
void bubblesort (void *base, unsigned int nmemb, unsigned int size,
                 int (*comp) (const void *, const void *)) {
  unsigned int i, j;
  for (i = nmemb - 1; i > 0; i--)
    for (j = 0; j < i; j++)
      if (comp ((char *) base + j * size, (char *) base + (j + 1) * size) > 0)
        swap ((char *) base + j * size, (char *) base + (j + 1) * size, size);
}
```



Bubblesort - swap

- **`swap`** has to swap the contents of two areas of memory, independently of their type
- Simple implementation is presented below

```
void swap (void *a, void *b, unsigned int size) {
  char tmp;
  unsigned int i;
  for (i = 0; i < size; i++)
    {
      tmp = *((char *) a + i);
      *((char *) a + i) = *((char *) b + i);
      *((char *) b + i) = tmp;
    }
}
```

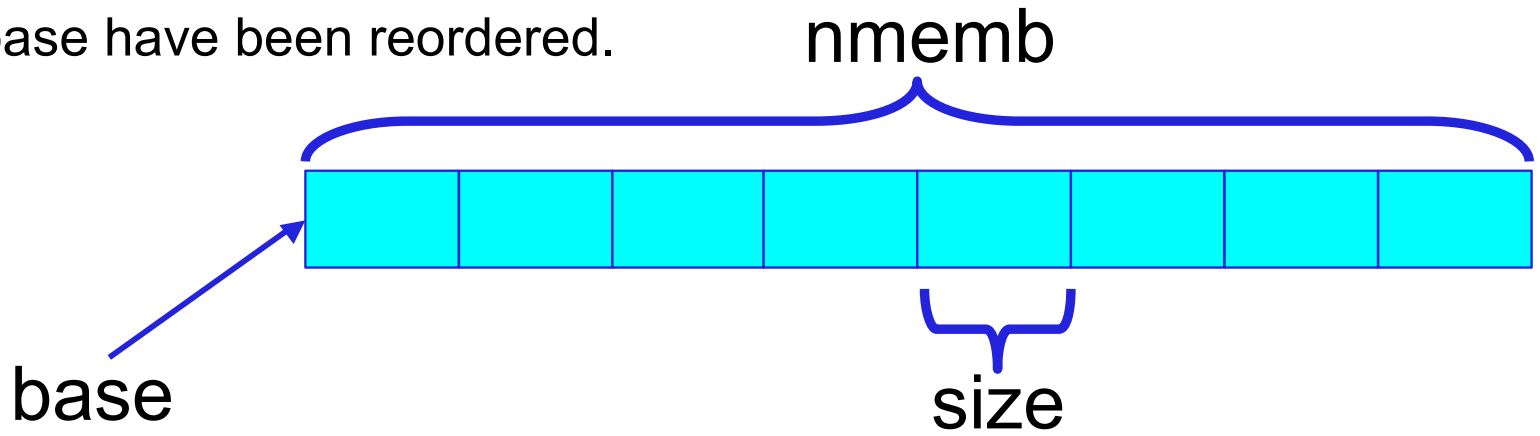
Bubblesort - Example

```
int compAccNo (const void *p1, const void *p2) {
  return ((CustAccount *) p1)->accNo - ((CustAccount *) p2)->accNo;
}

int compName (const void *p1, const void *p2) {
  return strcmp (((CustAccount *) p1)->name, ((CustAccount *) p2)->name);
}

int compNum (const void *p1, const void *p2) {
  return *((int *) p1) - *((int *) p2);
}
```

```
CustAccount accounts[] = { ... };
int numbers[] = { 15, 128, 3, 27, 32, 56, 1, 87 };

bubblesort (accounts, sizeof (accounts) / sizeof (*accounts),
            sizeof (*accounts), compName);

bubblesort (accounts, sizeof (accounts) / sizeof (*accounts),
            sizeof (*accounts), compAccNo);

bubblesort (numbers, sizeof (numbers) / sizeof (*numbers),
            sizeof (*numbers), compNum);
```

qsort

- `void qsort(void * base, size_t nmemb, size_t size, int (* compar)(const void *, const void *));`
  - **`qsort`** sorts an array (beginning at base) of **`nmemb`** objects.
  - **`size`** describes the size of each element of the array.
  - You must supply a pointer to a comparison function, using the argument **`compar`**
  - This permits sorting objects of unknown properties.
    - **`compar`** should accept two arguments, each a pointer to an element of the array
    - The result of **`compar`** must be negative if the first argument is less than the second, zero if the two arguments match, and positive if the first argument is greater than the second (where *less than* and *greater than* refer to what ever arbitrary ordering is appropriate).
  - The array is sorted in place; that is, when **`qsort`** returns. the array elements beginning at base have been reordered.

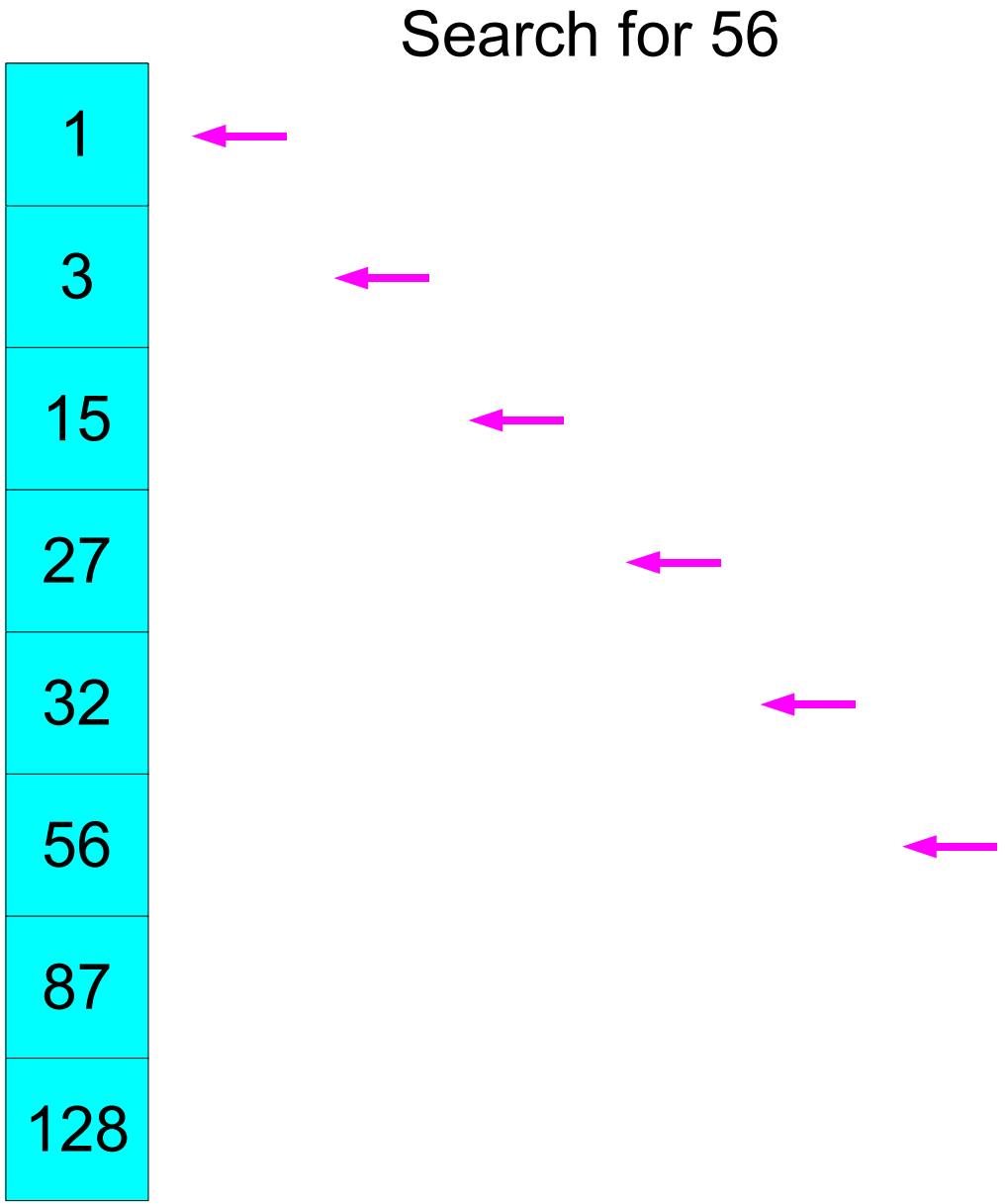


qsort example

```
typedef struct {
  char name[21];
  float balance;
  int accNo;
} CustAccount;

CustAccount accounts[] = { ... };

int compAccNo (const void *p1, const void *p2) {
  return ((CustAccount *) p1)->accNo - ((CustAccount *) p2)->accNo;
}

int compName (const void *p1, const void *p2) {
  return strcmp (((CustAccount *) p1)->name, ((CustAccount *) p2)->name);
}

int main()
{
  ...

  /* sort by name */
  qsort (accounts, sizeof (accounts) / sizeof (*accounts), sizeof (*accounts),
         compName);


  /* sort by accNo */
  qsort (accounts,
         sizeof (accounts) /
         sizeof (*accounts), sizeof (*accounts), compAccNo);
  ...

}
```

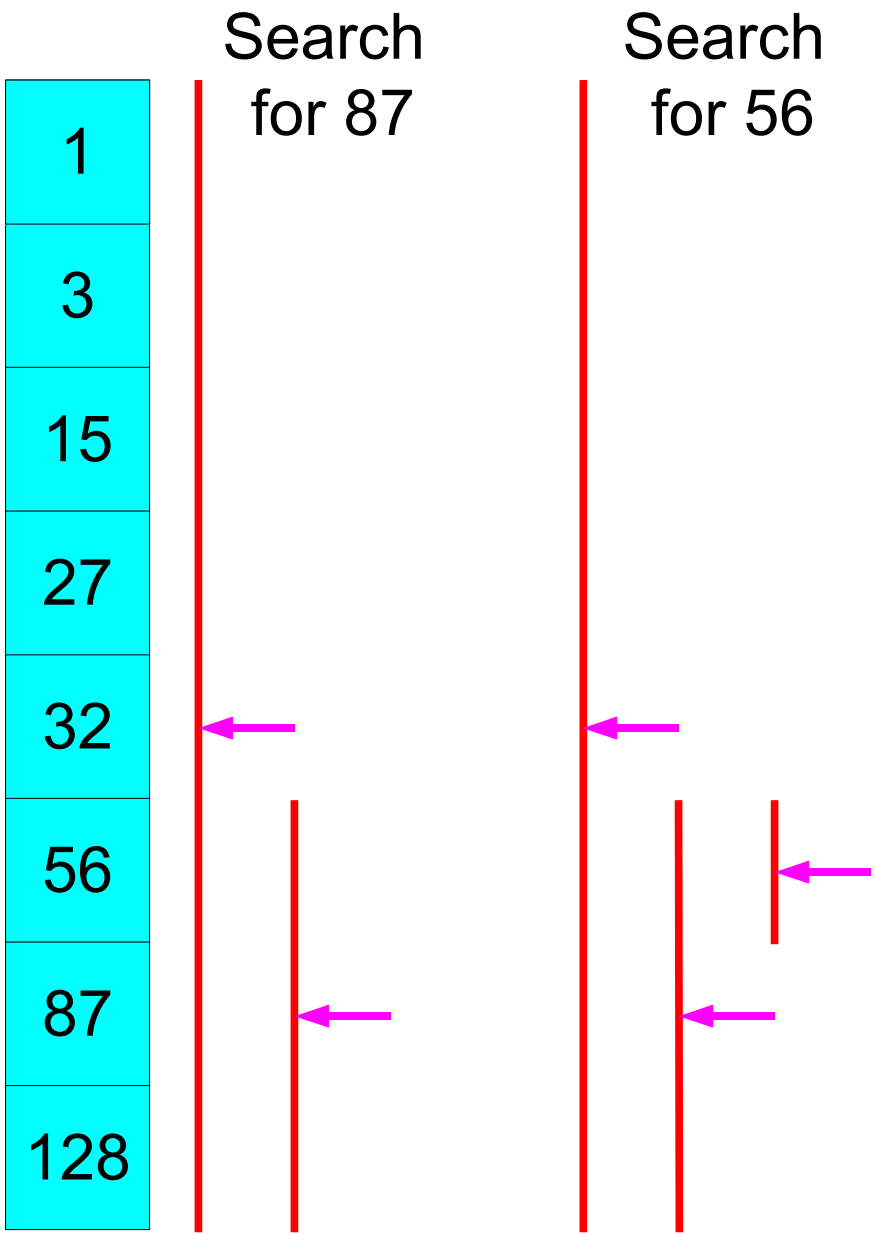
Linear Search

- Start from beginning of an array and compare every element one by one



Binary Search

- Allows for fast searching in the sorted array
- We start in the middle and then determine in which half to search further
- Apply again until key found or remaining area is empty



Binary Search - C Code

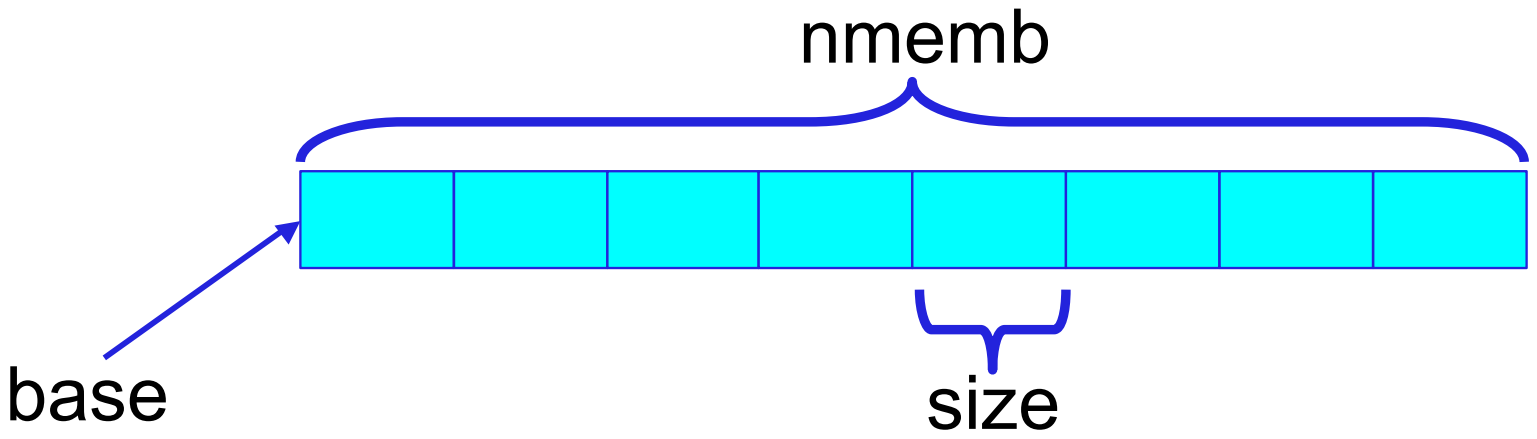
```
int * binarySearch (int what, int *low, int *high) {
  while (high > low)
    {
      int *middle = low + (high - low) / 2;
      if (*middle == what)
        return middle;
      else if (*middle > what)
        high = middle;
      else
        low = middle + 1;
    }
  return NULL;
}

void search (int number, int *low, int *high) {
  if (binarySearch (number, low, high))
    printf ("%d found\n",number);
  else
    printf ("%d not found\n",number);
}

int main () {
  int numbers[] = { 1, 3, 15, 27, 32, 56, 87, 128 };
  search (87, numbers, numbers + sizeof (numbers) / sizeof (*numbers));
  search (56, numbers, numbers + sizeof (numbers) / sizeof (*numbers));
  search (16, numbers, numbers + sizeof (numbers) / sizeof (*numbers));
  return 0;
}
```

Pointer to the location just after the last element

Pointer to first element of the array

bsearch

- `void *bsearch(const void *key, const void *base,`
  `size_t nmemb, size_t size,`
  `int (*compar)(const void *, const void *));`
  - **`bsearch`** searches an array of **`nmemb`** objects, the initial member of which is pointed to by **`base`**, for a member that matches the object pointed to by **`key`**. The size of each member of the array is specified by **`size`**.
  - The contents of the array should be in ascending sorted order according to the comparison function referenced by **`compar`**.
  - **`bsearch`** returns a pointer to a matching member of the array, or **`NULL`** if no match is found. If there are multiple elements that match the key, the element returned is unspecified.



bsearch example

```
typedef struct {
  char name[21];
  float balance;
  int accNo;
} CustAccount;

CustAccount accounts[SIZE];

int compAccNo (const void *p1, const void *p2) {
  return ((CustAccount *) p1)->accNo - ((CustAccount *) p2)->accNo;
}

int compName (const void *p1, const void *p2) {
  return strcmp (((CustAccount *) p1)->name, ((CustAccount *) p2)->name);
}

CustAccount * findByName (const char *name) {
  CustAccount reference;
  strcpy (reference.name, name);
  return bsearch (&reference, accounts, sizeof (accounts) / sizeof (*accounts),
                  sizeof (*accounts), compName);
}

CustAccount * findByAccNo (int accNo) {
  CustAccount reference;
  reference.accNo = accNo;
  return bsearch (&reference, accounts, sizeof (accounts) / sizeof (*accounts),
                  sizeof (*accounts), compAccNo);
}
```