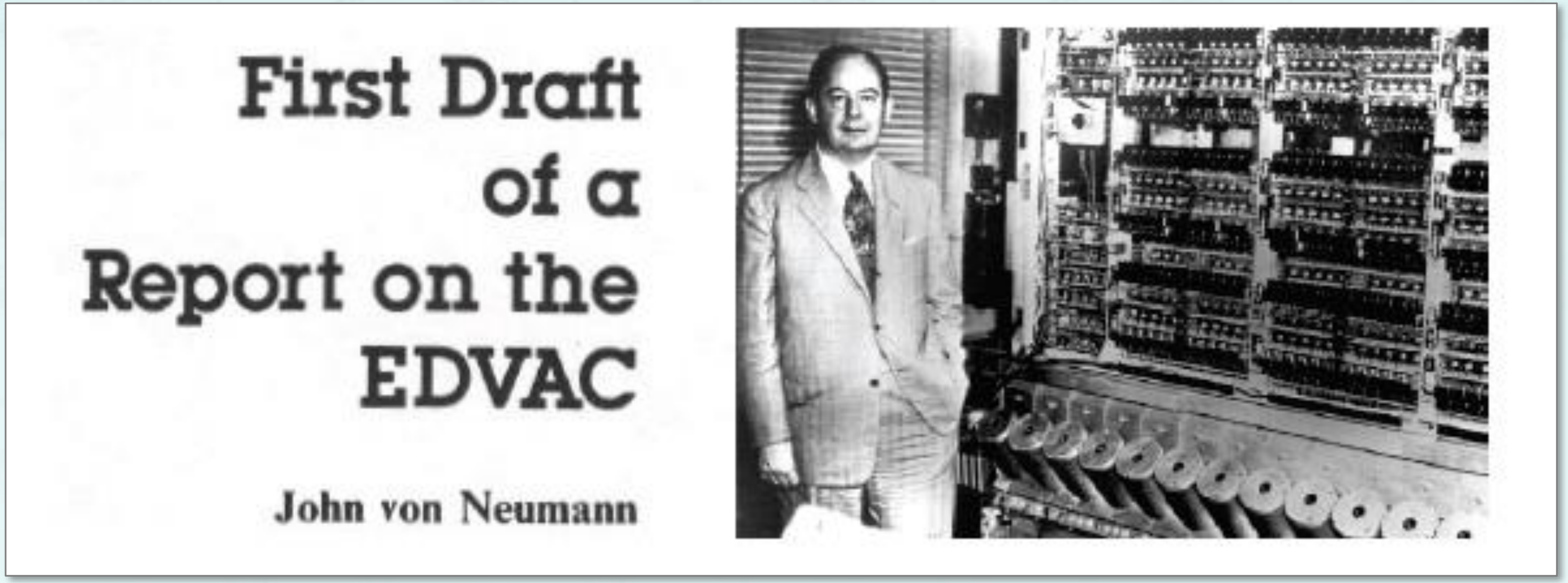
Data Structures
Chapter 1

1. **Recursion**
   - Recursion
   - **Mergesort**
2. Performance Analysis
3. Asymptotic Analysis

Mergesort

- Divide and conquer algorithm
- Recursive or non-recursive(Iteration) implementation
- It was implemented on the first general purpose computer and is still running.

the first general purpose computer and its inventor,



Mergesort: Algorithm

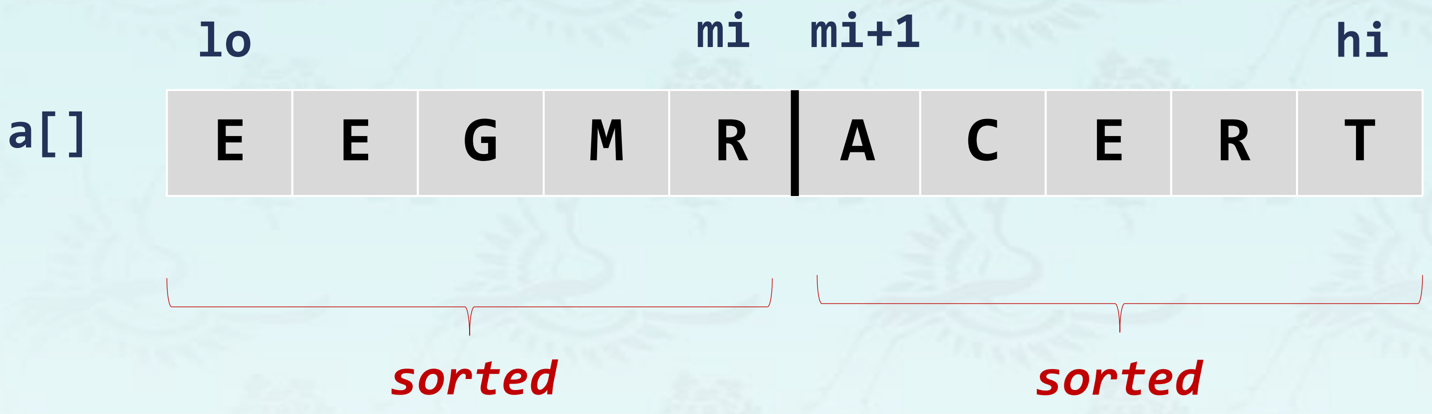
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves.

input	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
sort left half	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
sort right half	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge results	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Mergesort overview

Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.

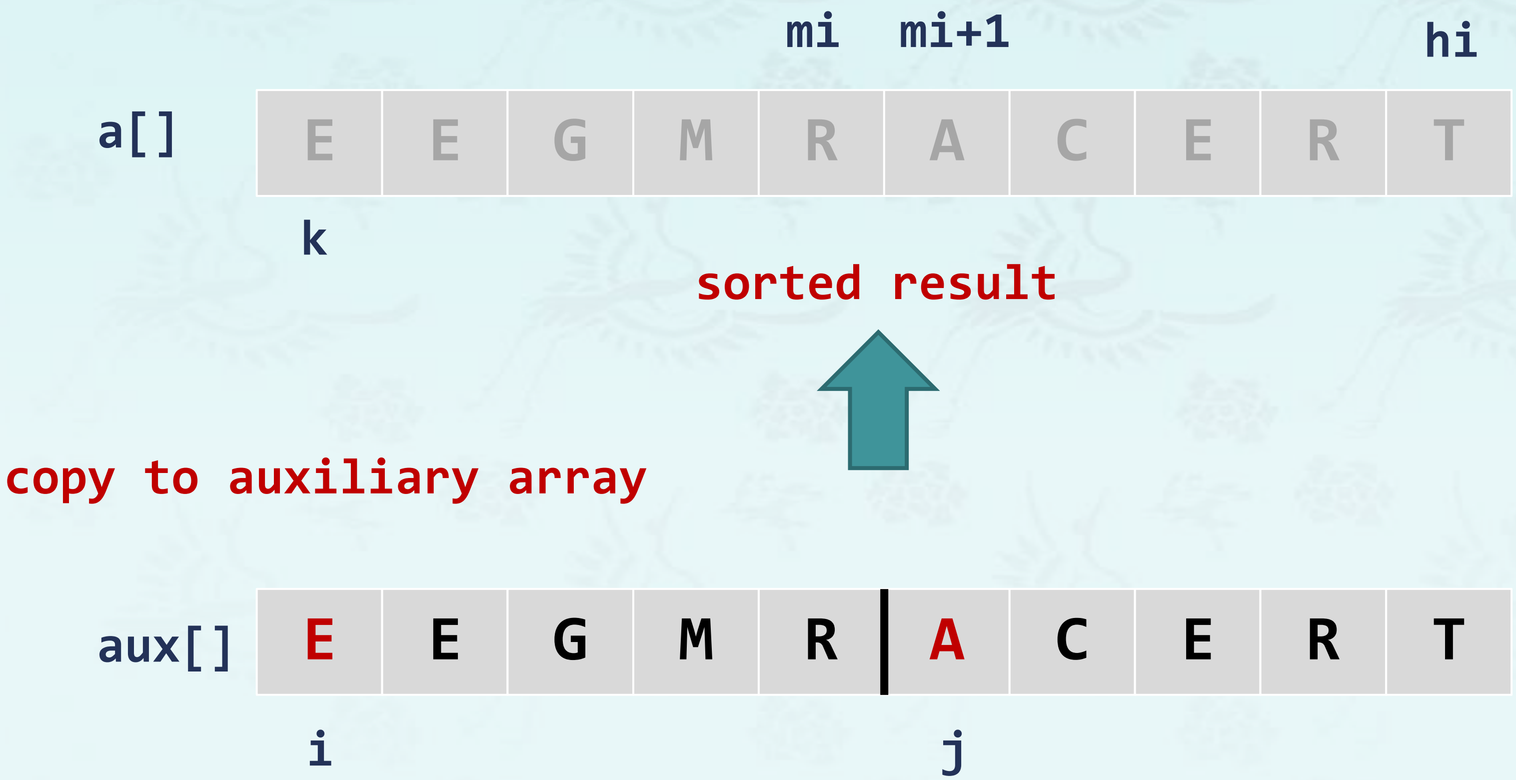
	lo				mi	mi+1				hi
a[]	E	E	G	M	R	A	C	E	R	T

copy to auxiliary array

aux[]	E	E	G	M	R	A	C	E	R	T

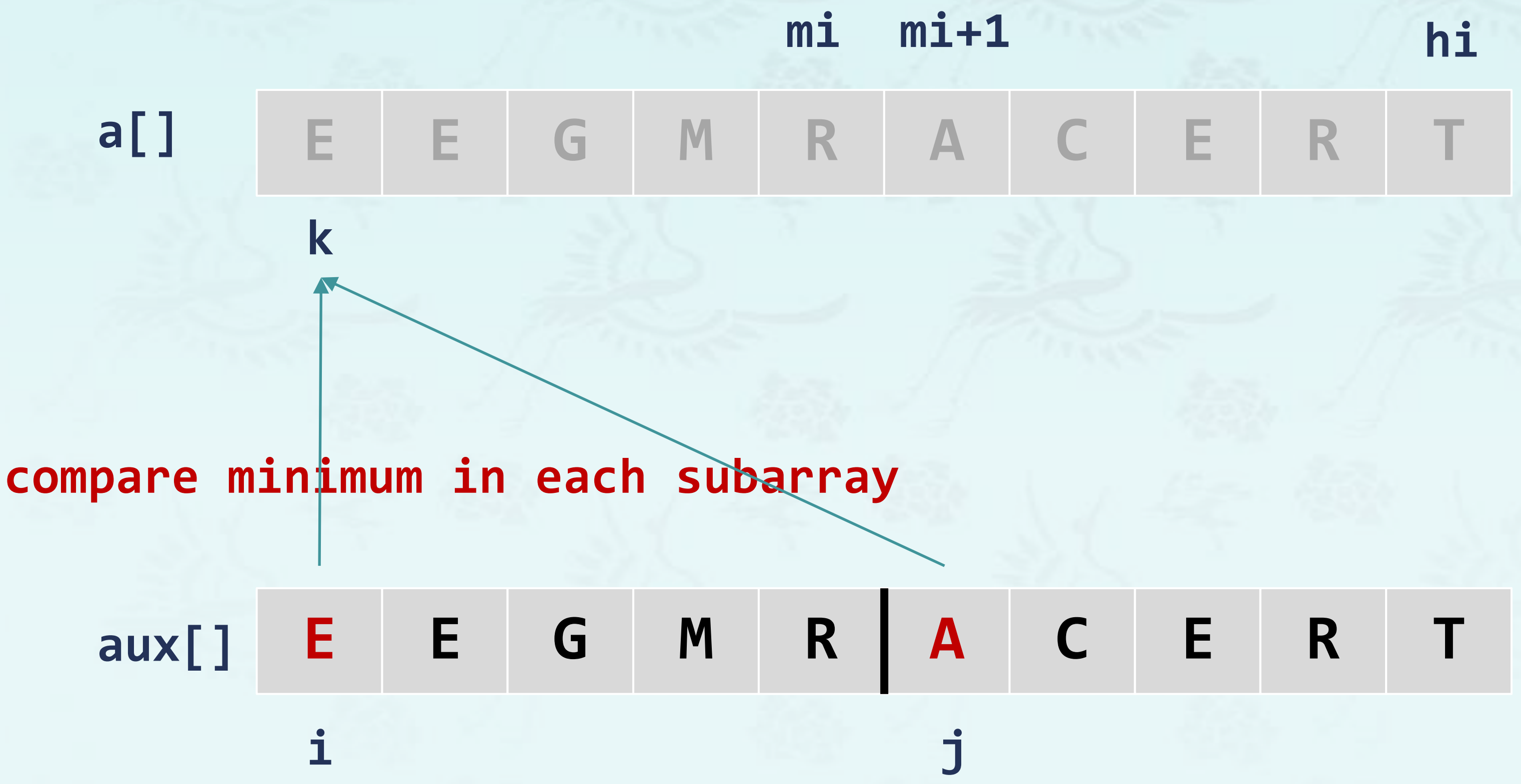
Mergesort: merge

- **Goal:** Given two sorted subarrays **a[lo]** to **a[mi]** and **a[mi+1]** to **a[hi]**, replace with sorted subarray **a[lo]** to **a[hi]**.



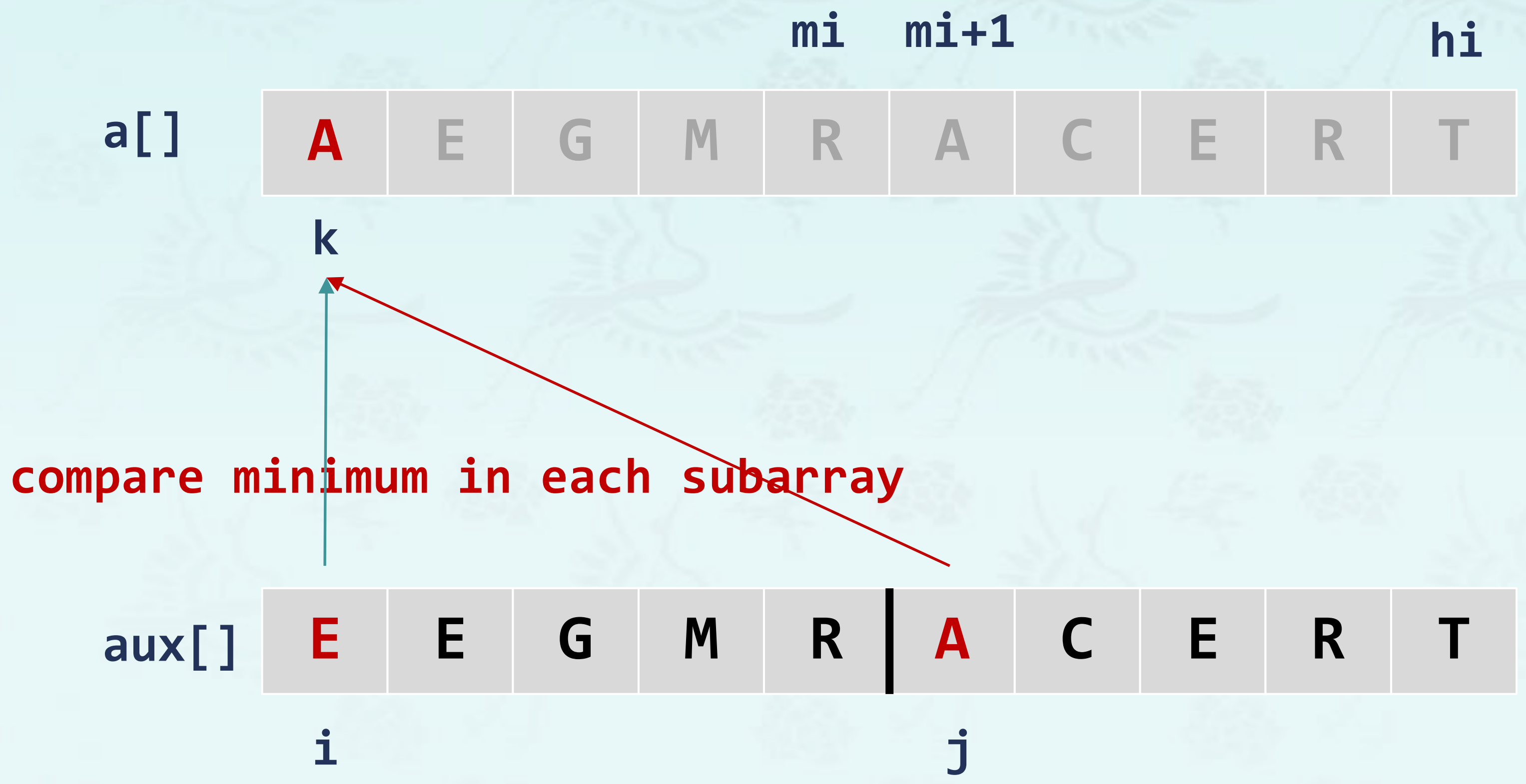
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



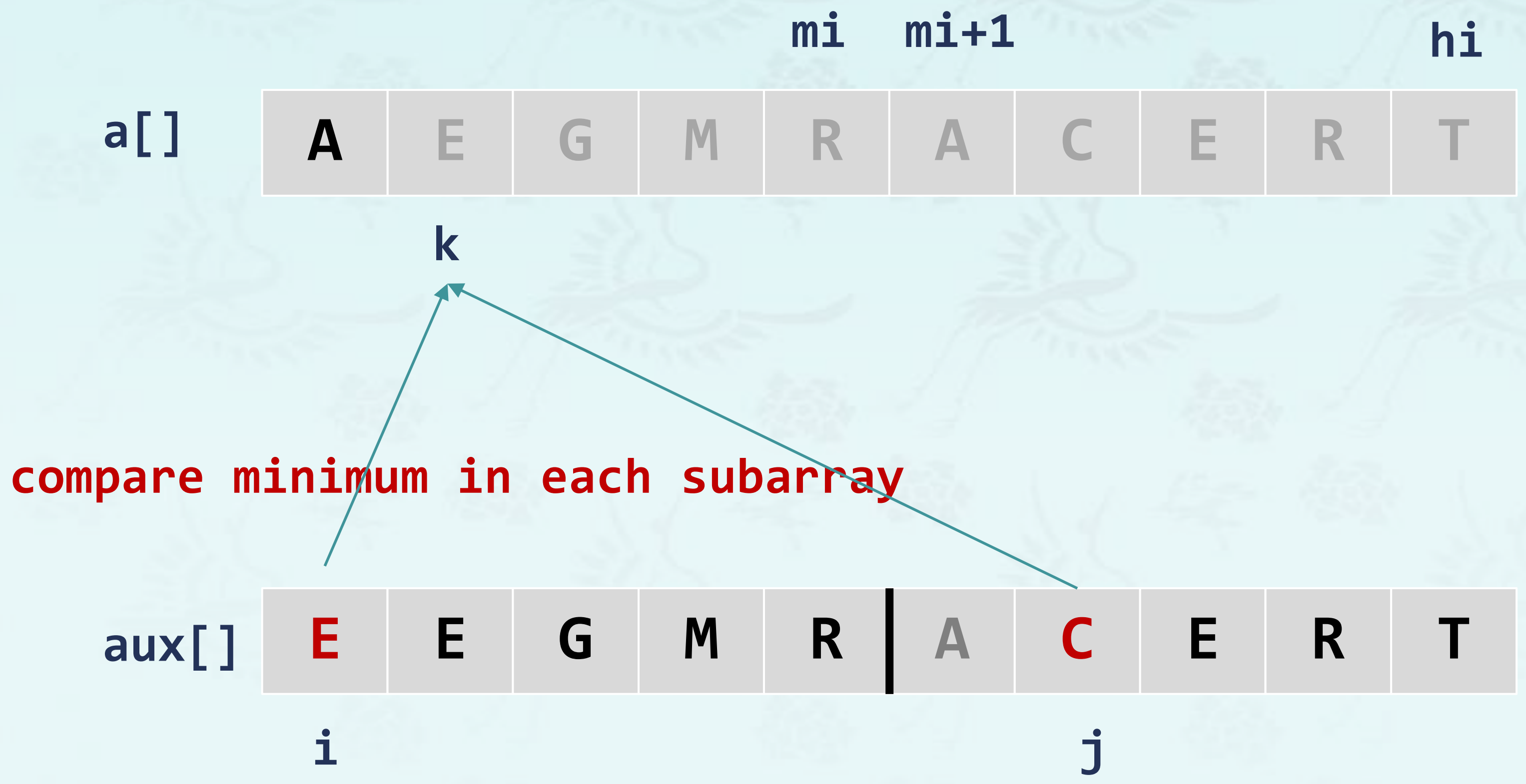
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



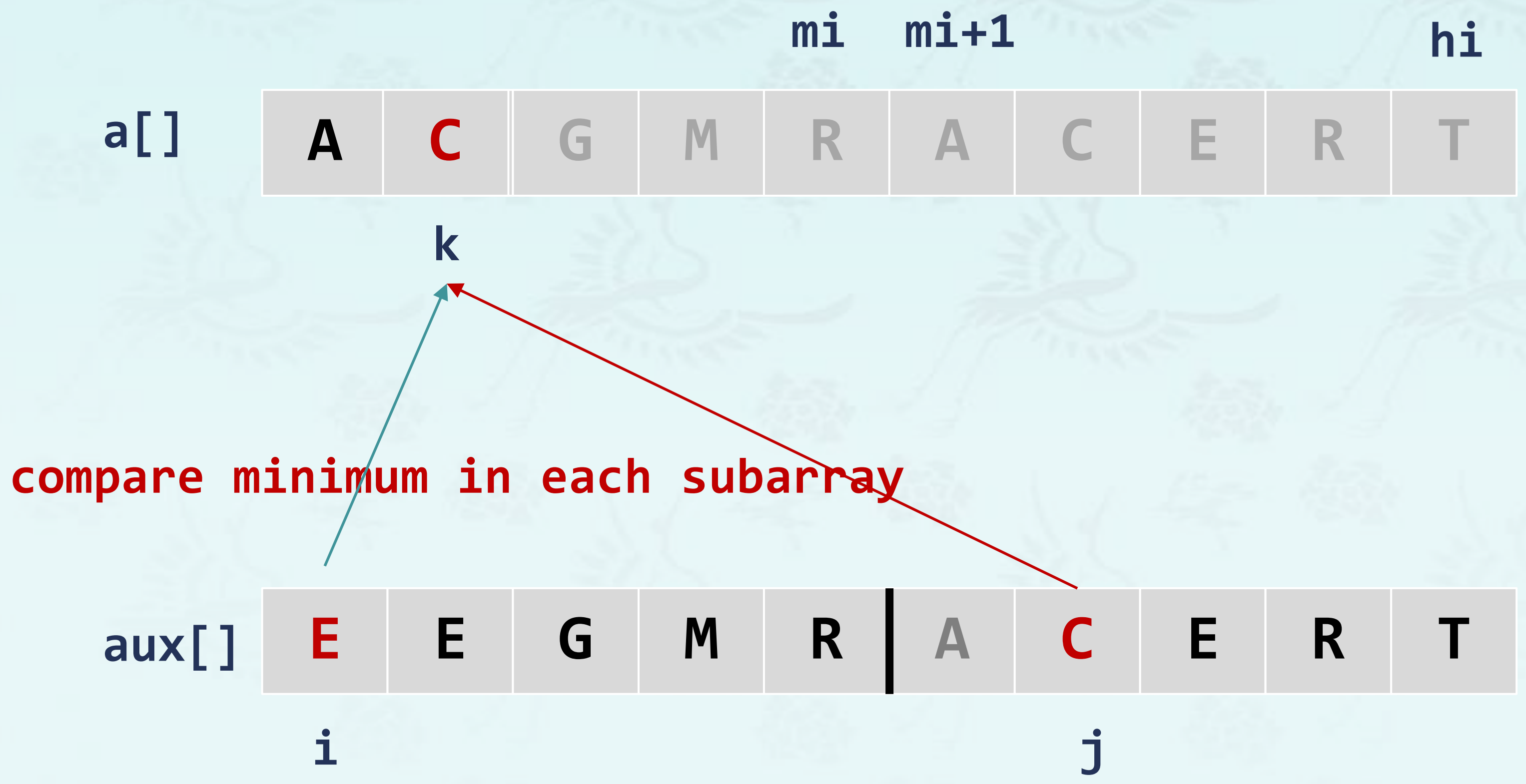
Mergesort: merge

- **Goal:** Given two sorted subarrays **a[lo]** to **a[mi]** and **a[mi+1]** to **a[hi]**, replace with sorted subarray **a[lo]** to **a[hi]**.



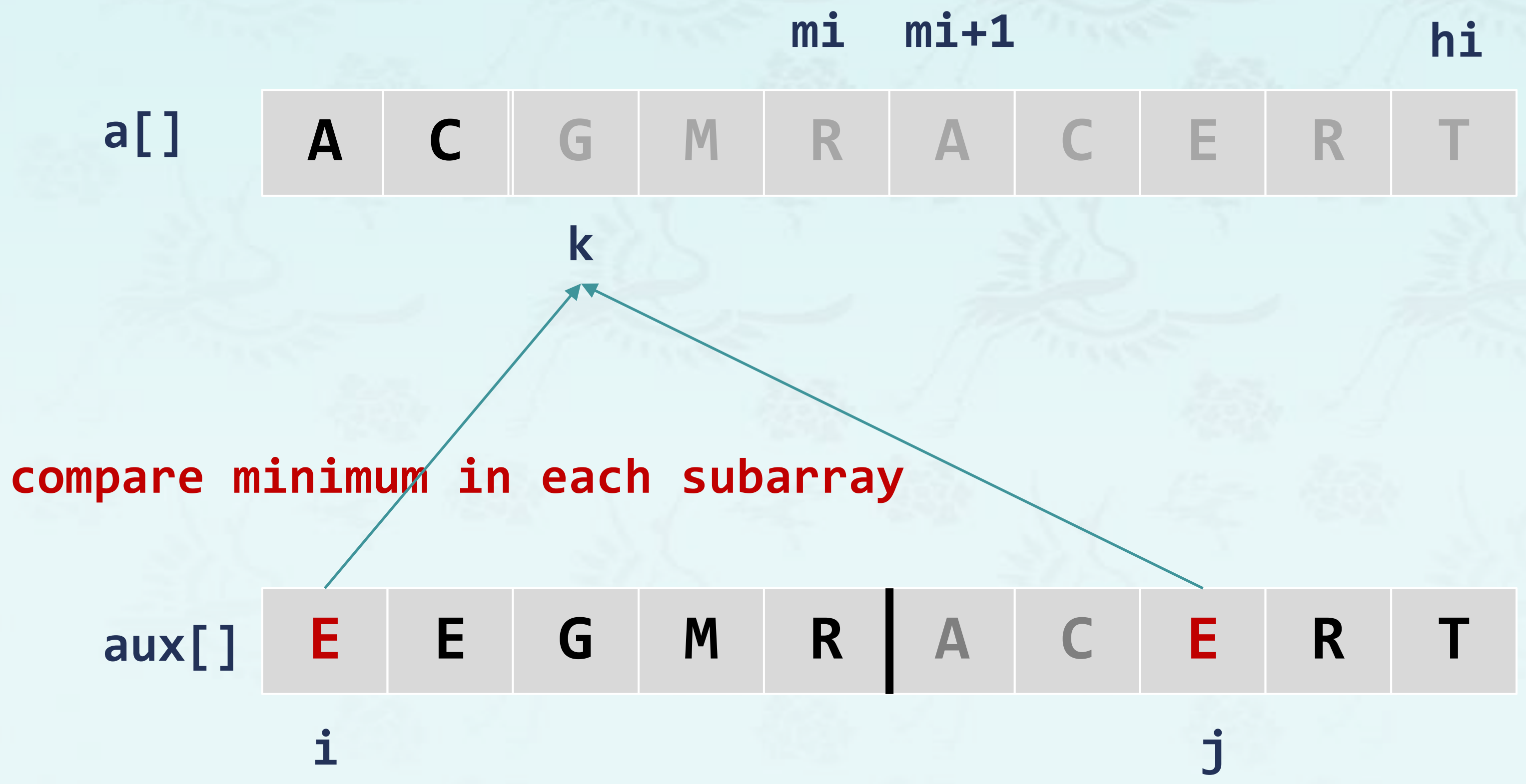
Mergesort: merge

- **Goal:** Given two sorted subarrays **a[lo]** to **a[mi]** and **a[mi+1]** to **a[hi]**, replace with sorted subarray **a[lo]** to **a[hi]**.



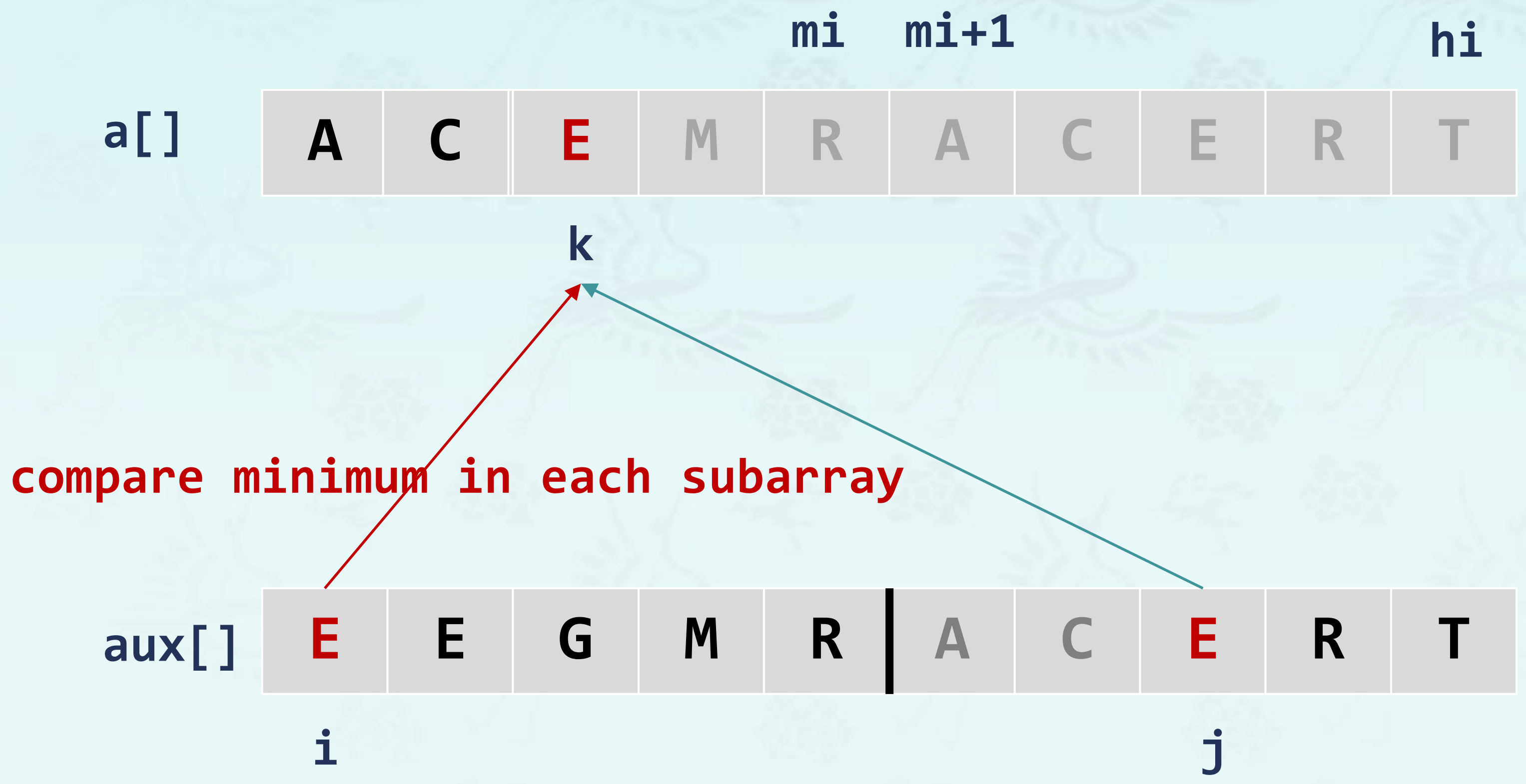
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



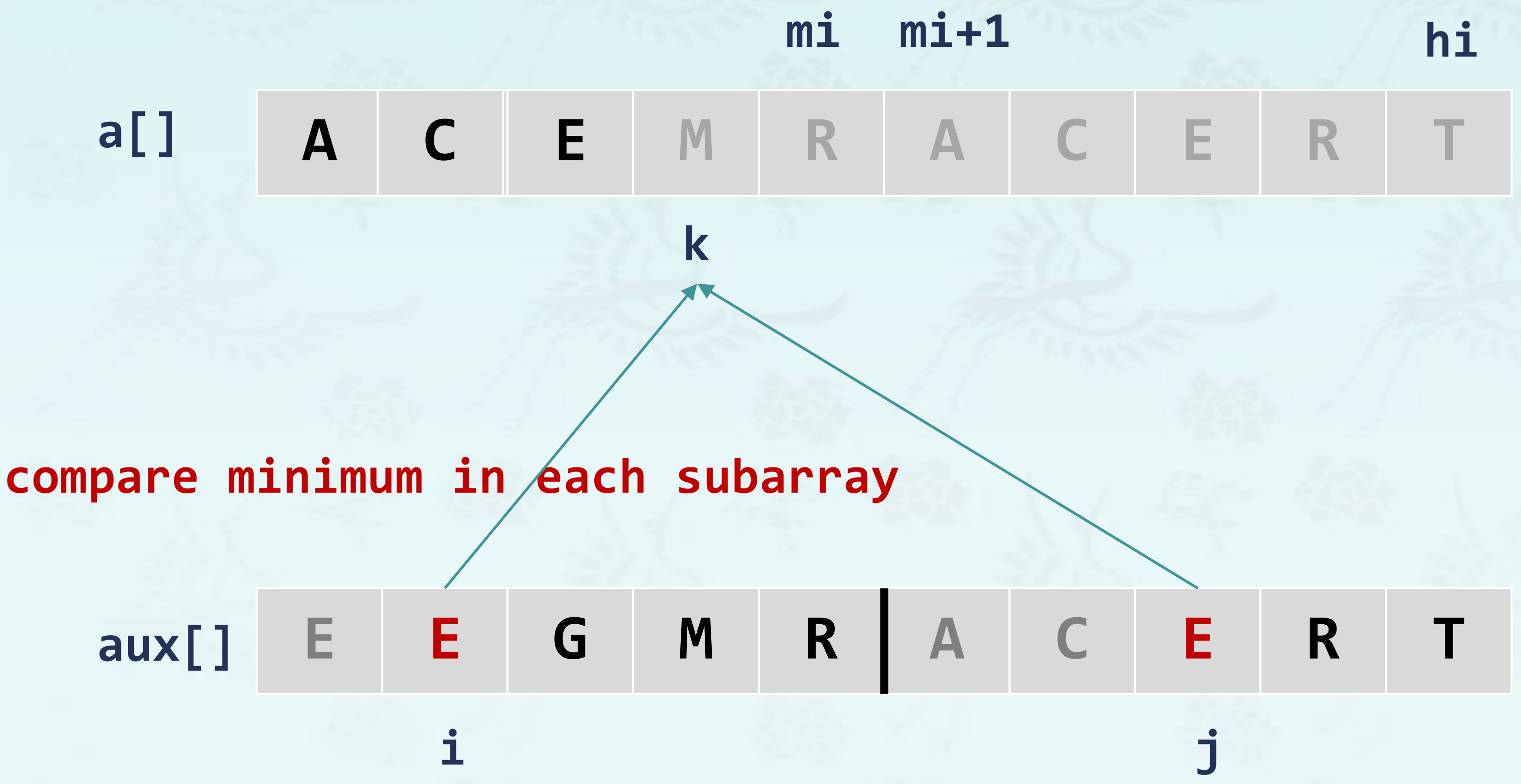
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



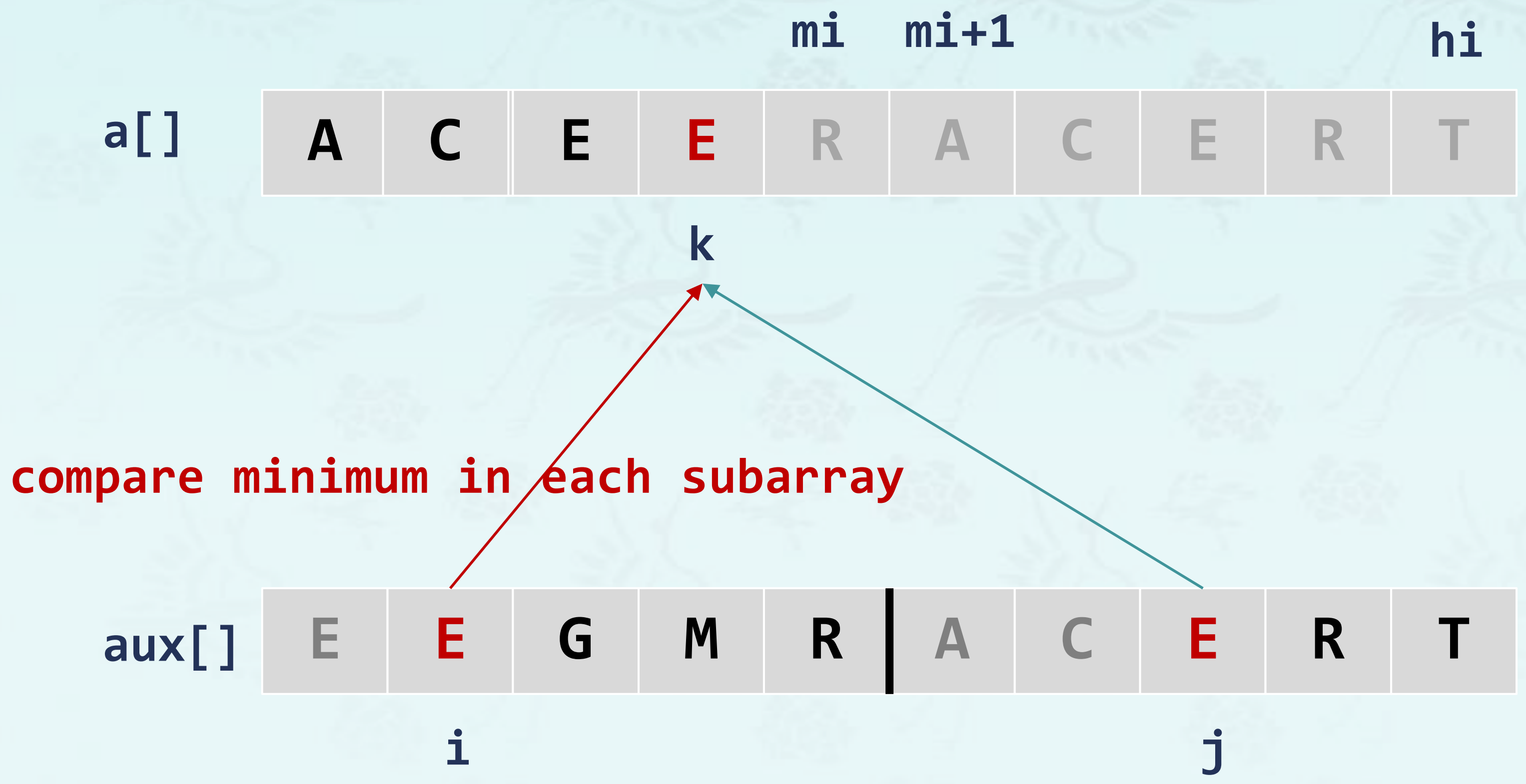
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



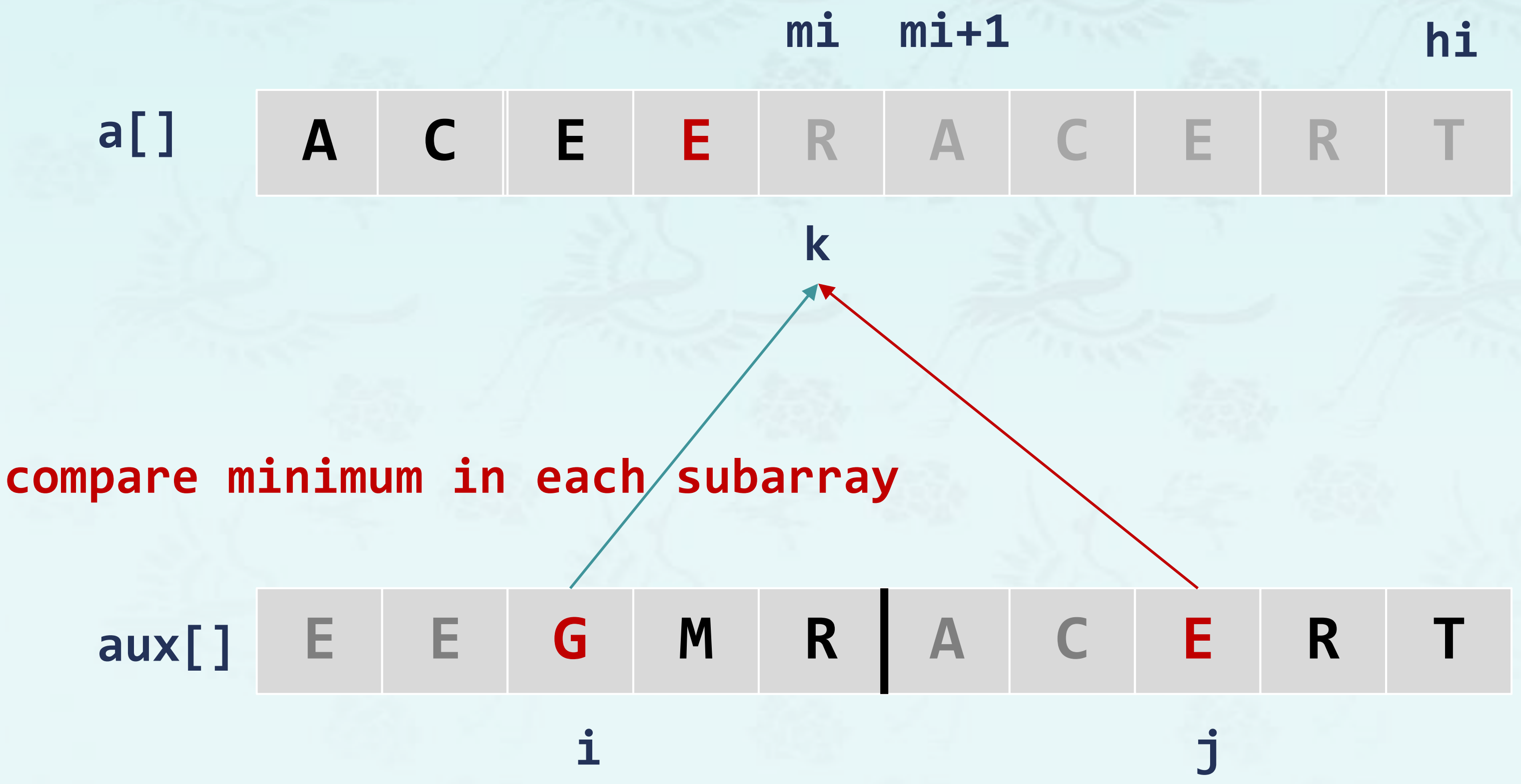
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



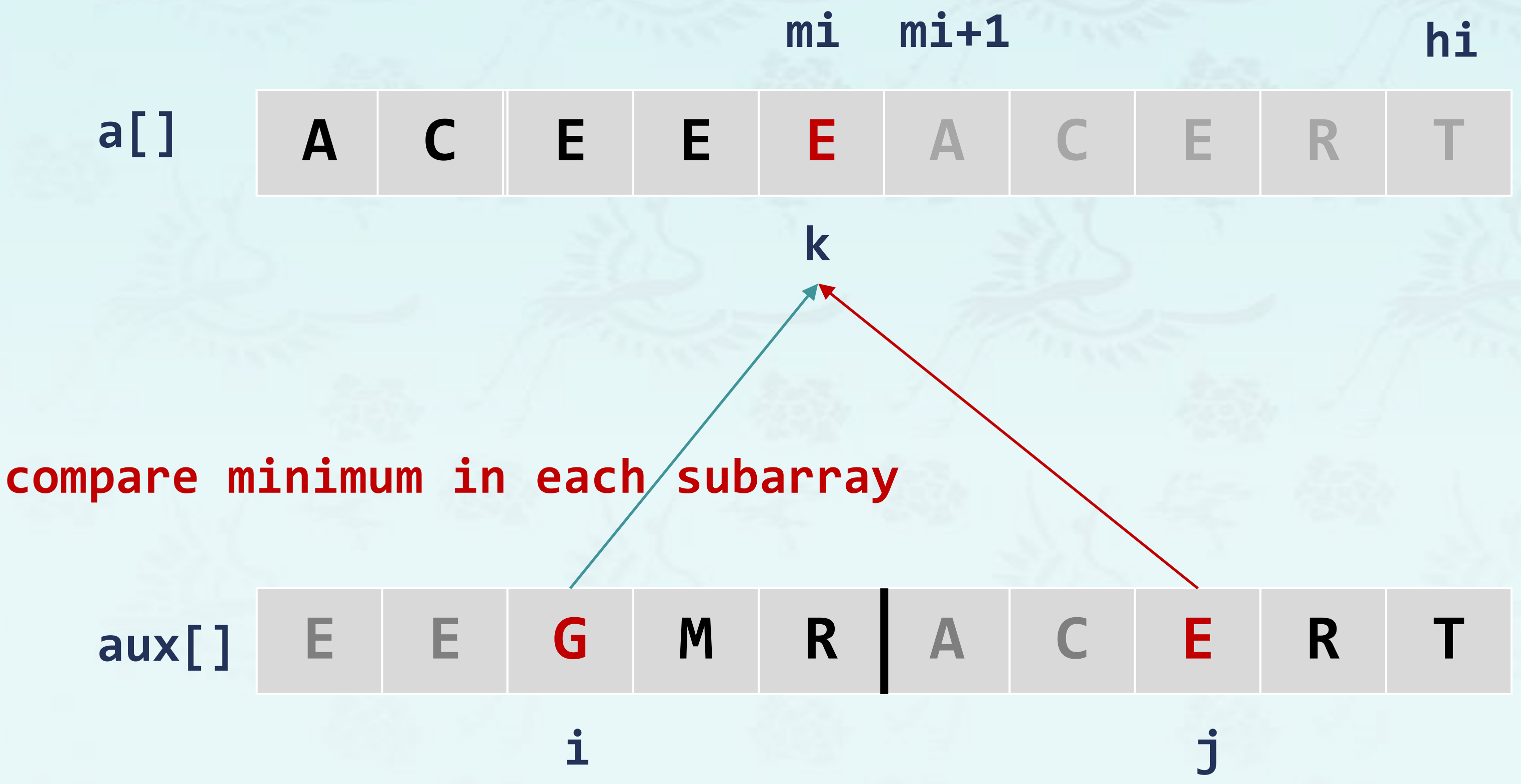
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



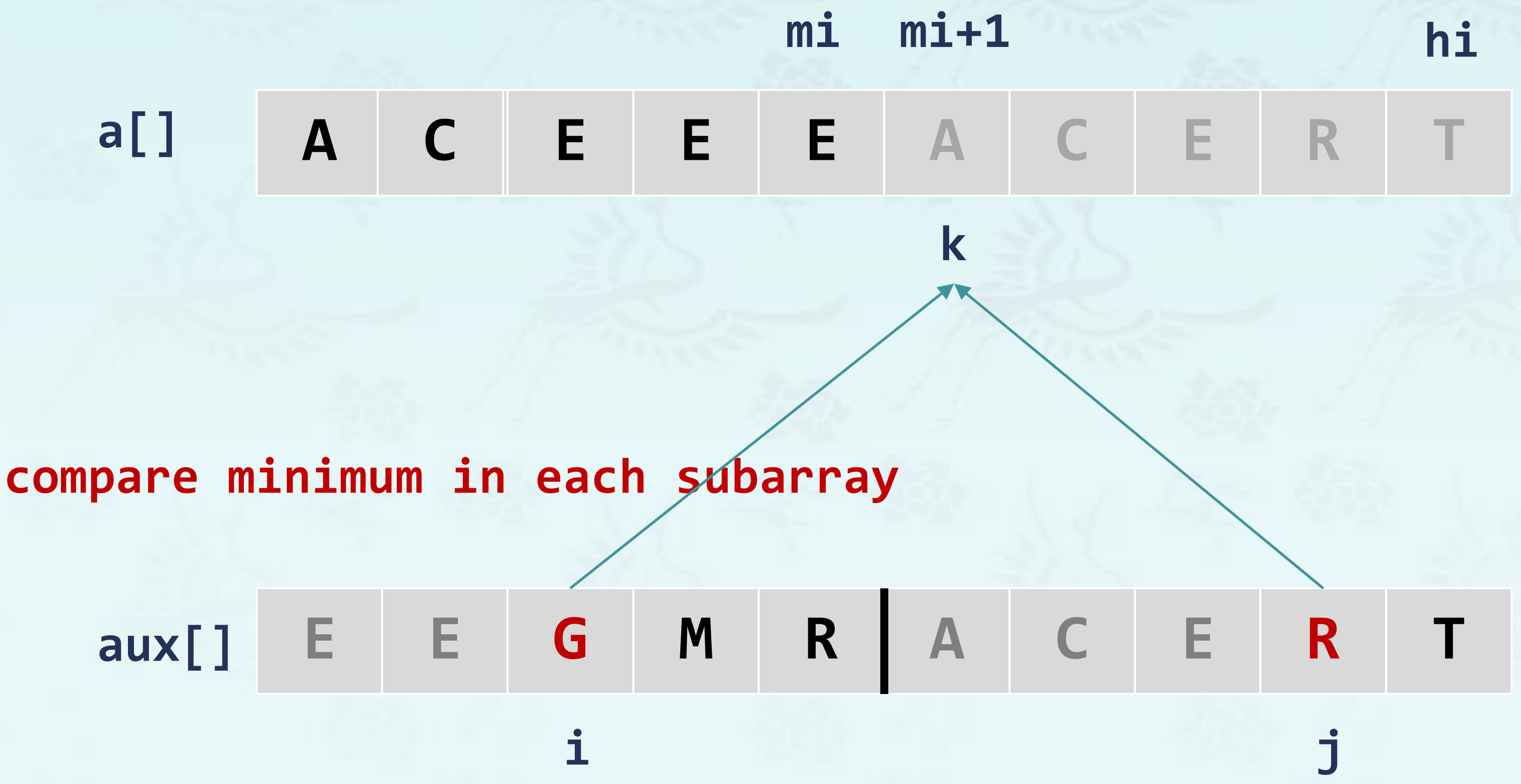
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



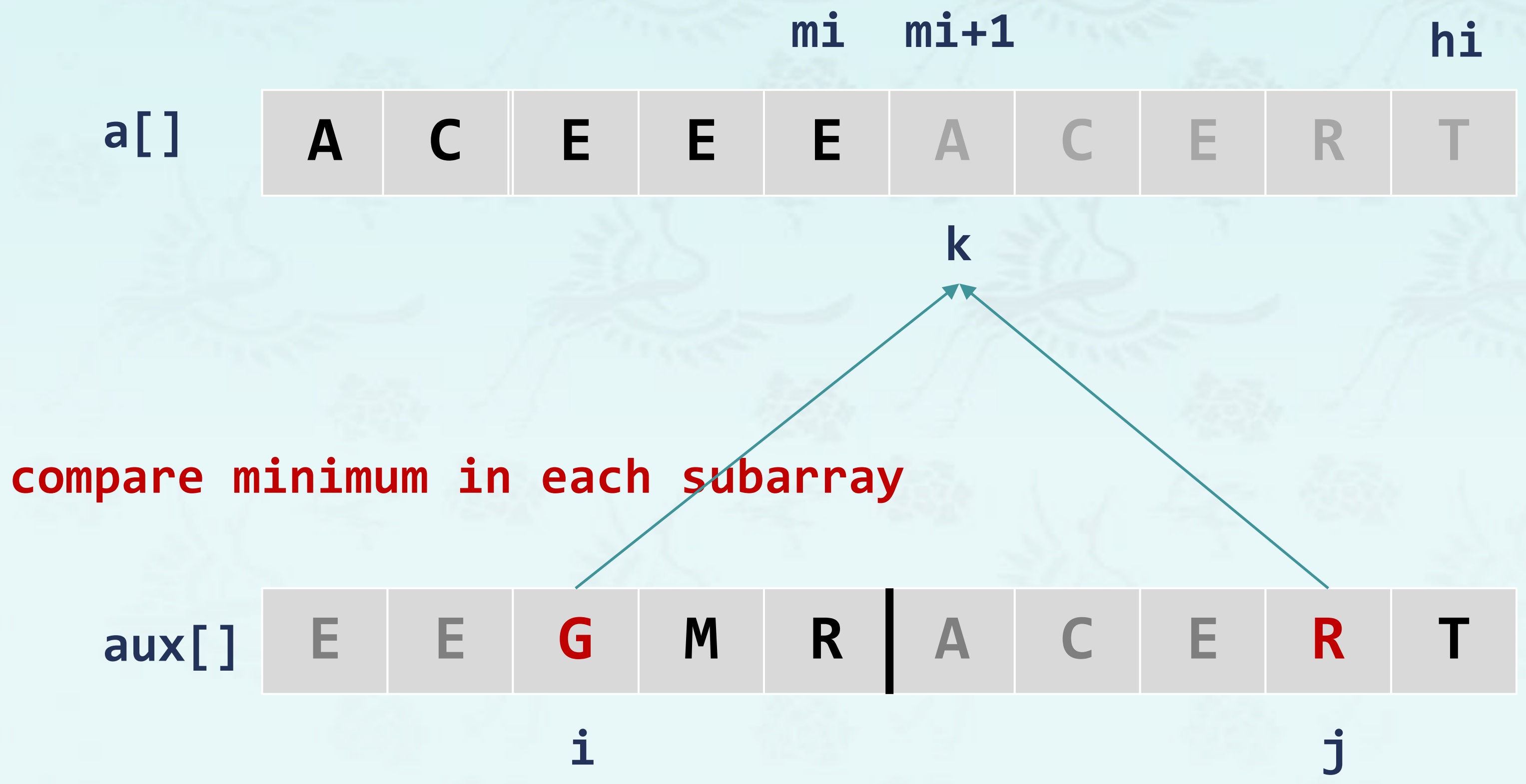
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



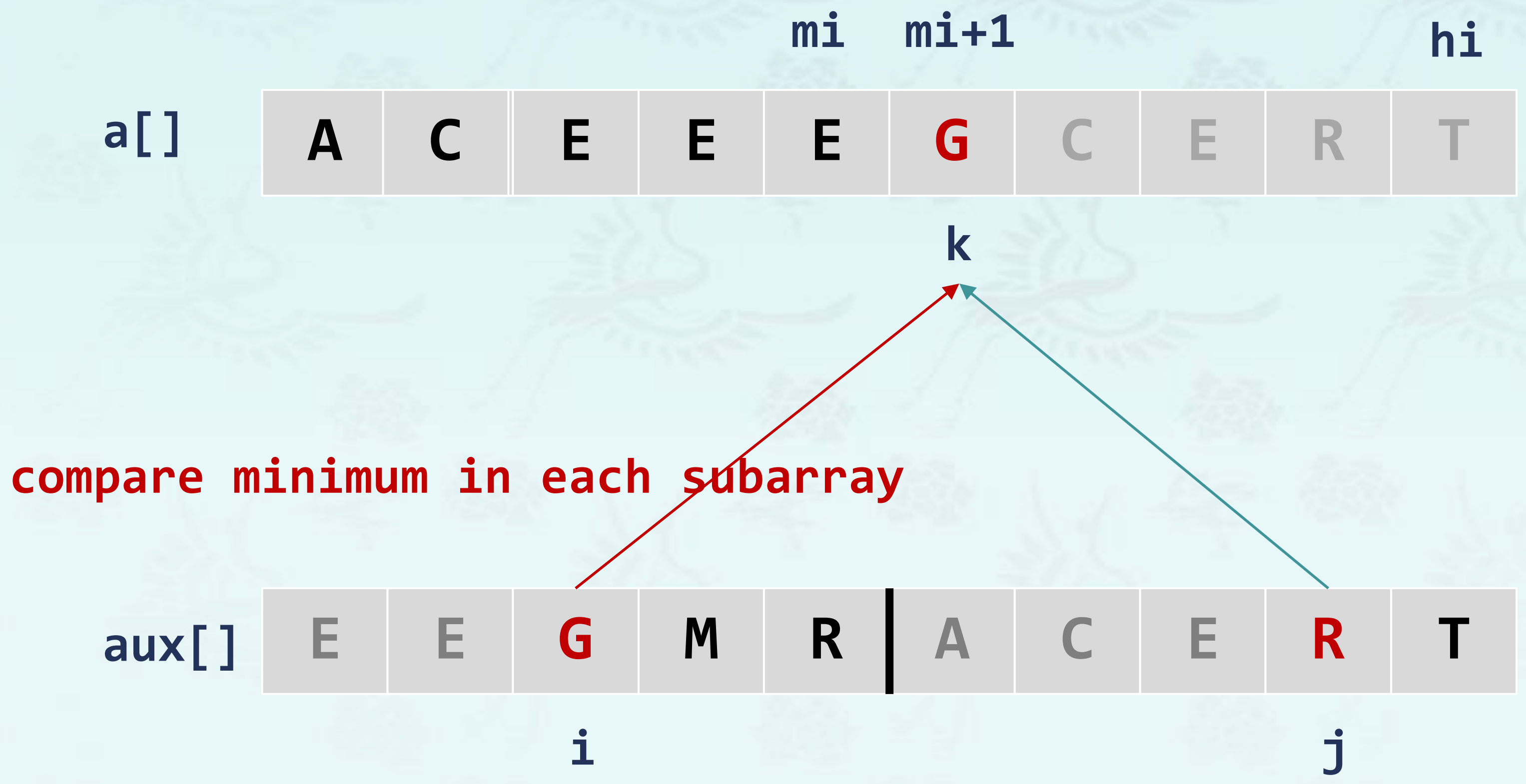
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



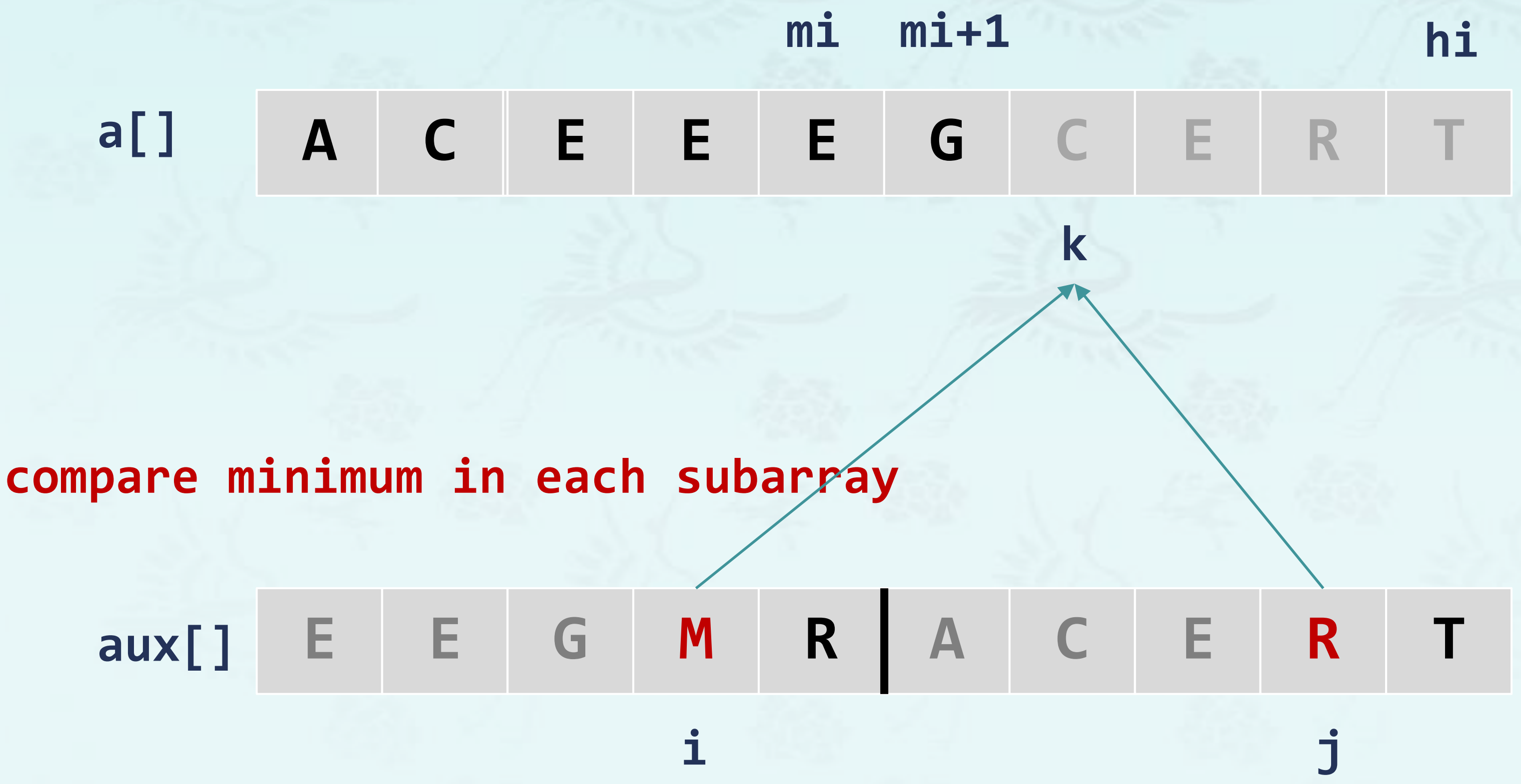
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



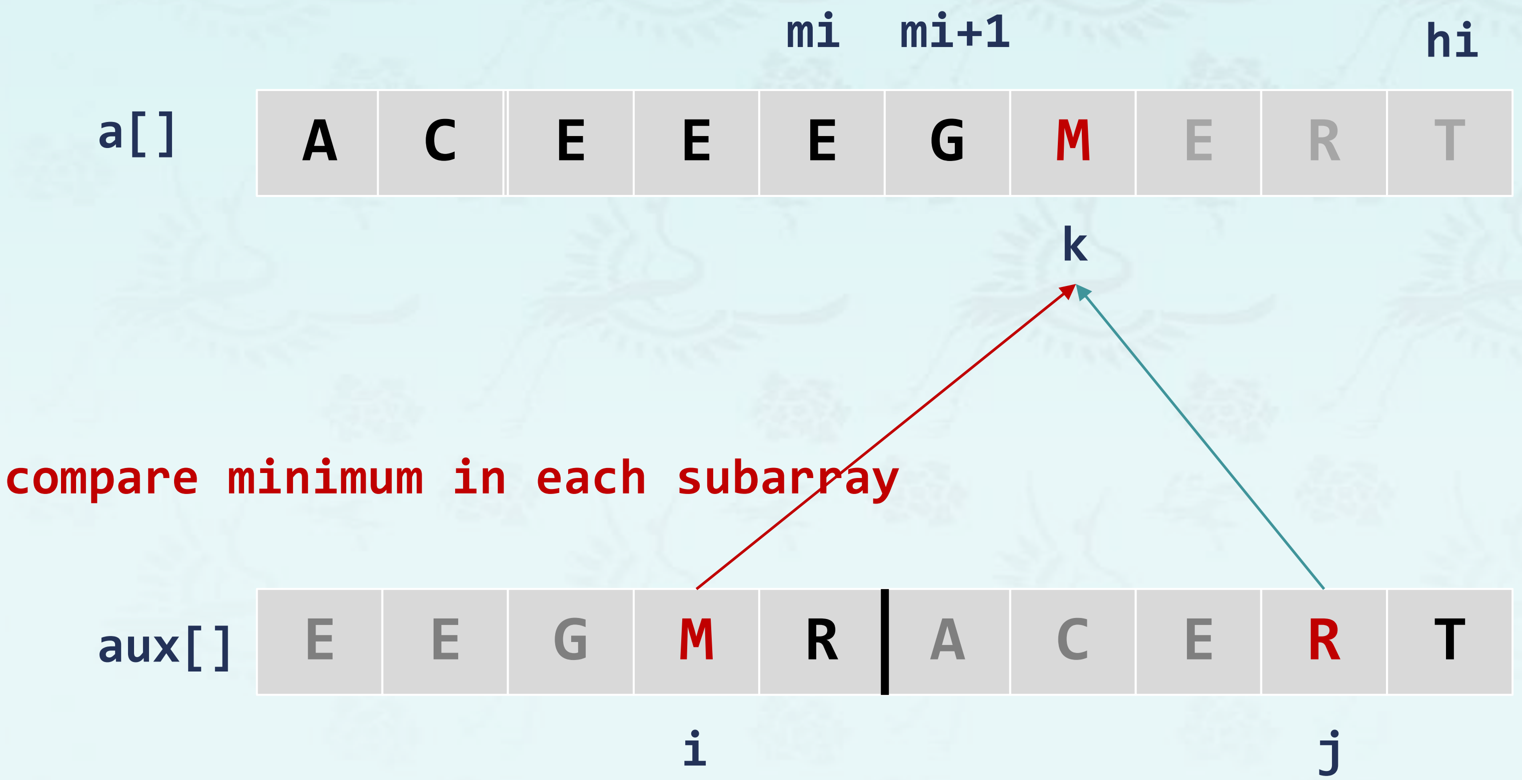
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



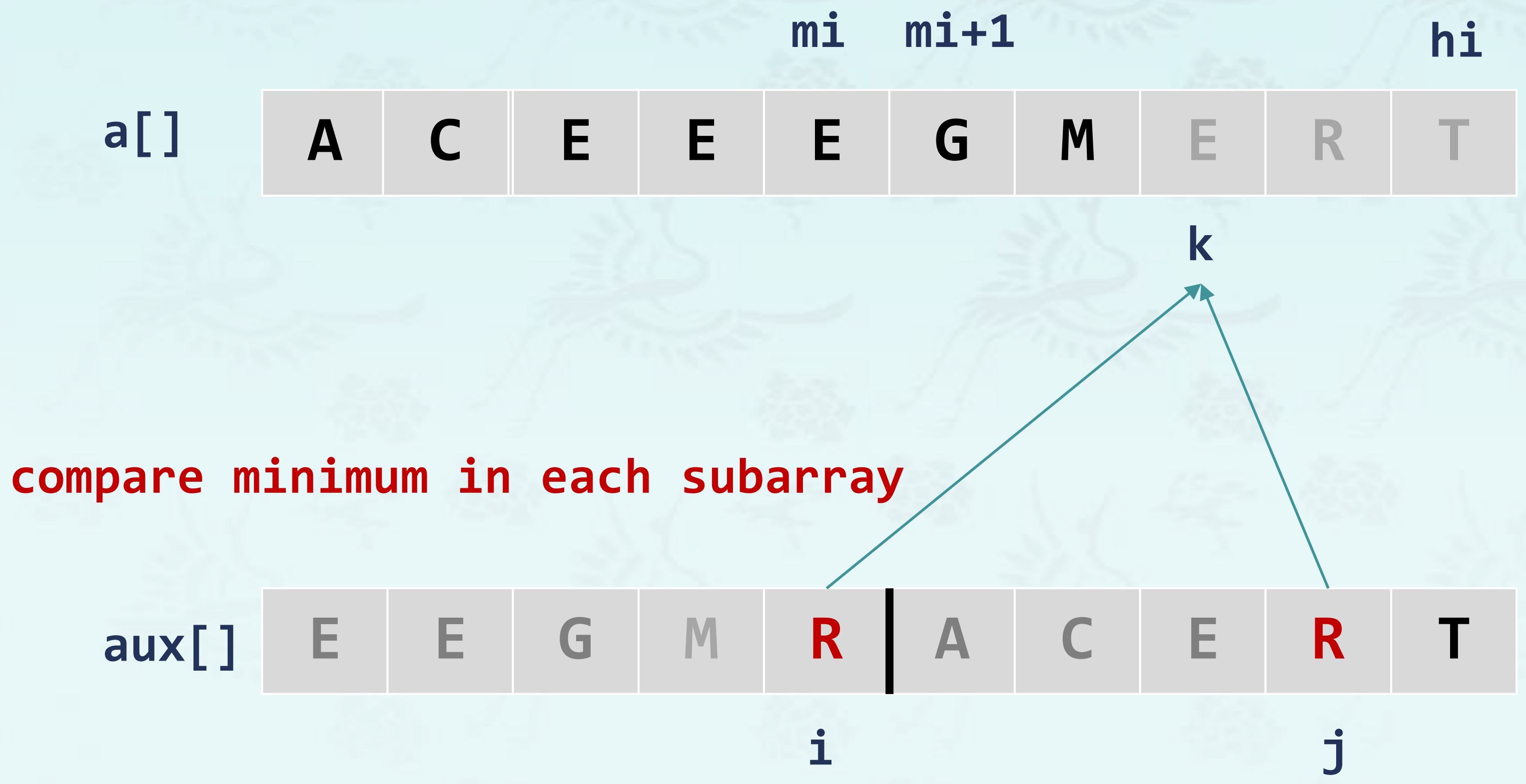
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



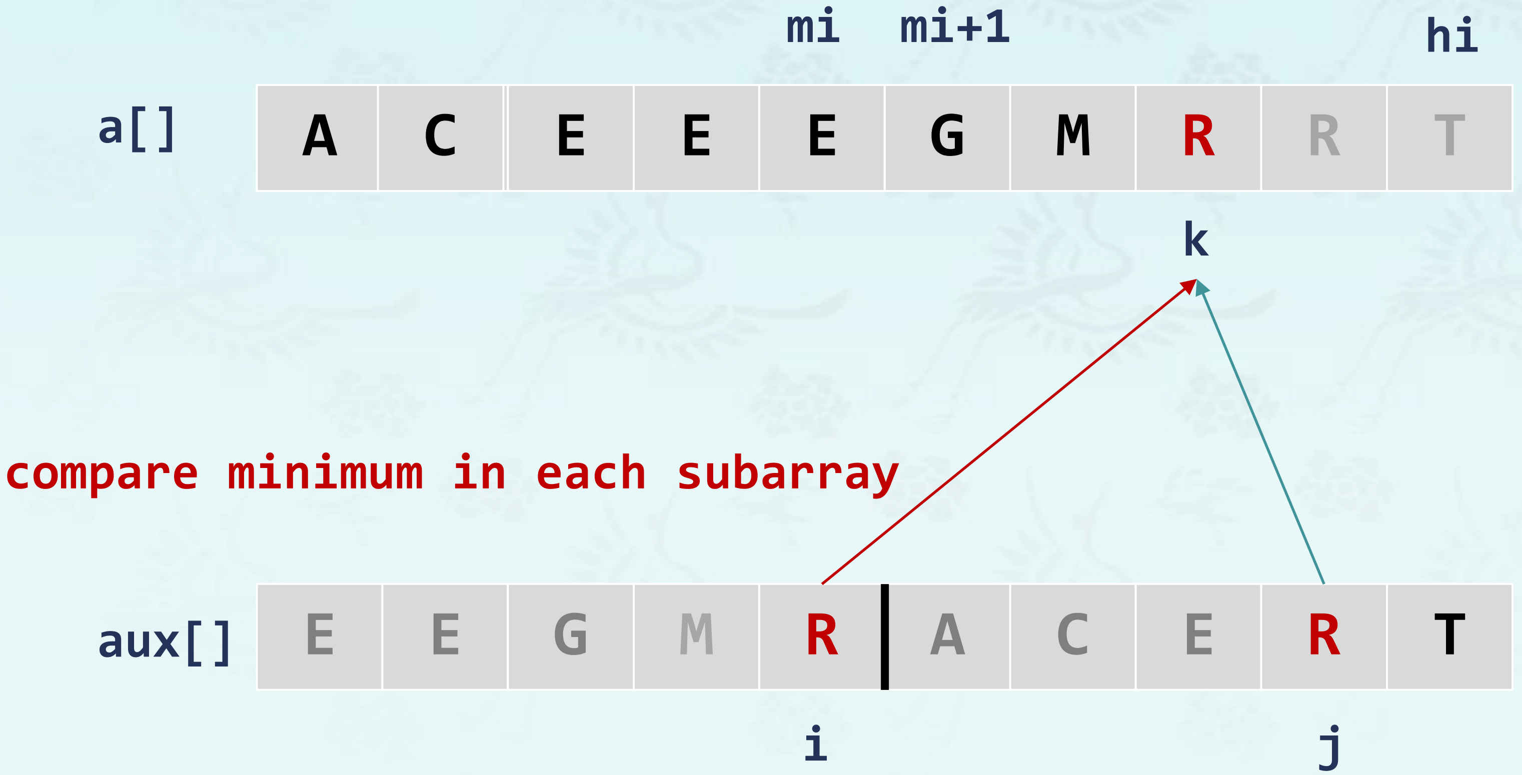
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



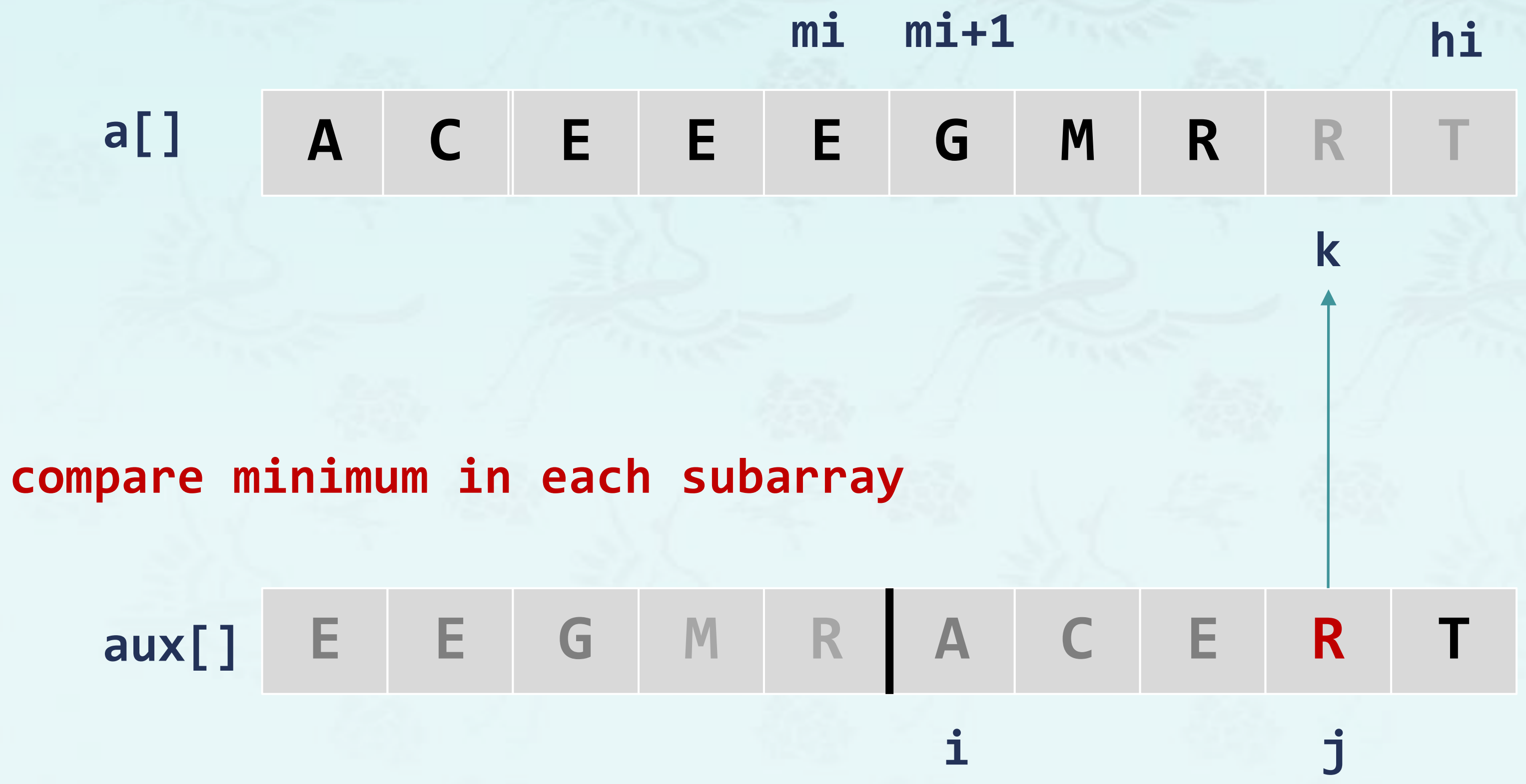
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



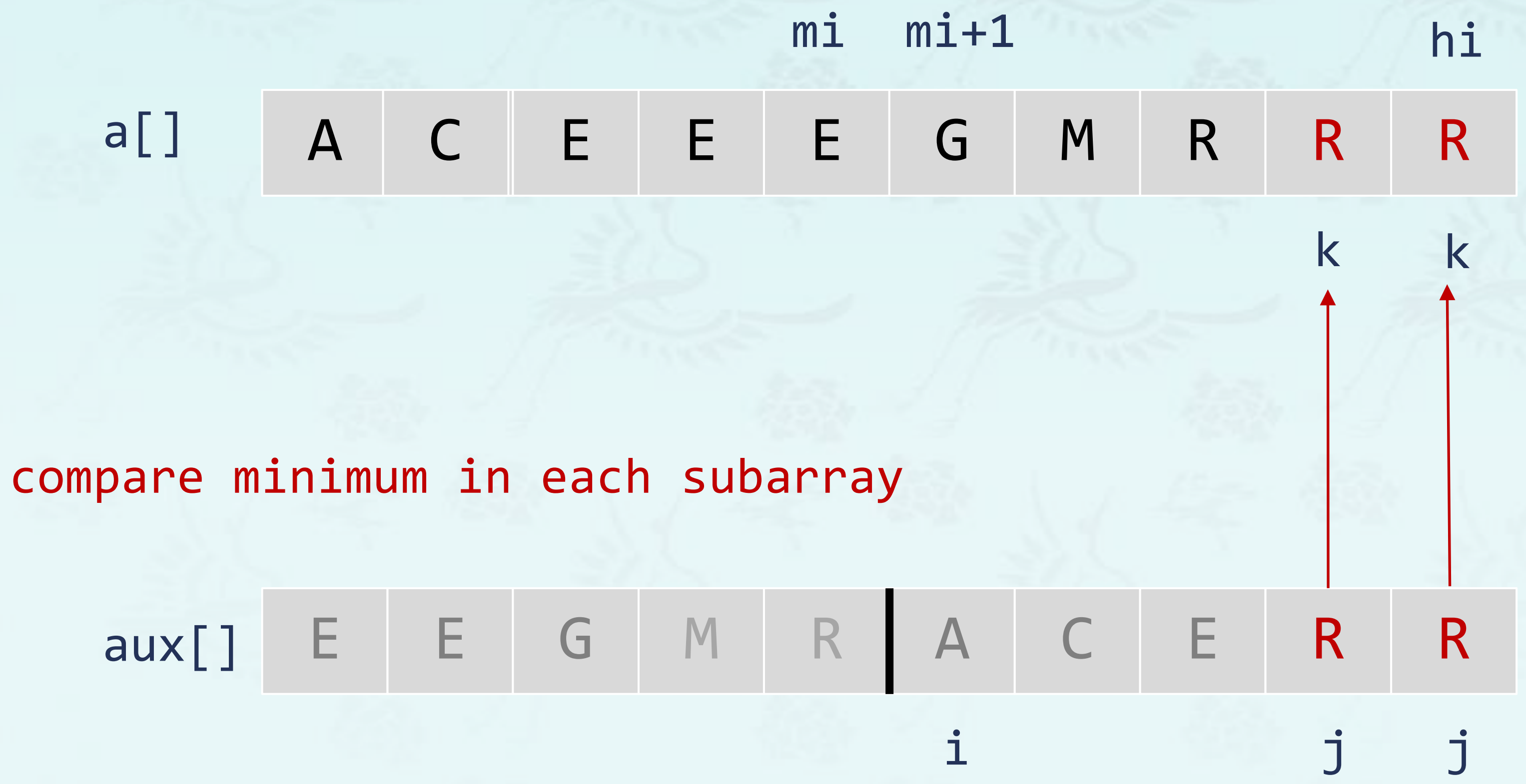
Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.



Mergesort: merge

- **Goal:** Given two sorted subarrays **`a[lo]`** to **`a[mi]`** and **`a[mi+1]`** to **`a[hi]`**, replace with sorted subarray **`a[lo]`** to **`a[hi]`**.

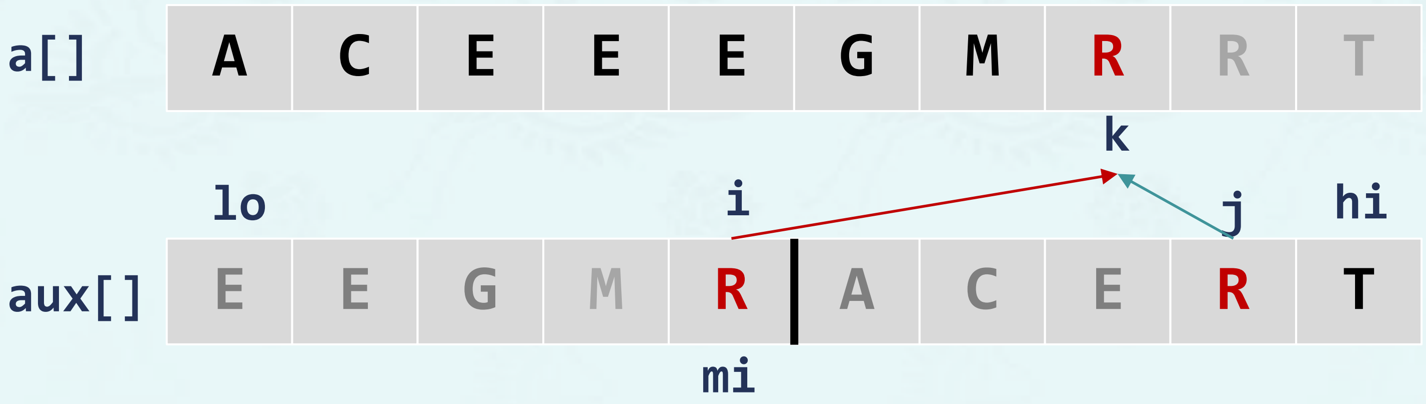
					mi	mi+1				hi
a[]	A	C	E	E	E	G	M	R	R	R

mergeSort complete using auxiliary array

aux[]	E	E	G	M	R	A	C	E	R	R

Mergesort: merge

- If your array is empty or has one element, it is sorted.
- If it has two elements, sort it by swapping as appropriate.
- If it has more than two elements, do this:
  - split the array in half at the midpoint **mi**;
  - call **merge sort** on the left half;
  - call **merge sort** on the right half;
  - **merge** the arrays by picking the smallest head element from the two sub-arrays until they are exhausted.



Example 5: Recursive binary search

- For instance, we want to search "23" from the array. If we find it, we return its array index; otherwise, -1 or something else.

0	1	2	3	4	5	6	7	8	9
2	5	8	9	16	23	31	56	62	71

lo=0	1	2	3	mi=4	5	6	7	8	hi=9
2	5	8	9	16	23	31	56	62	71

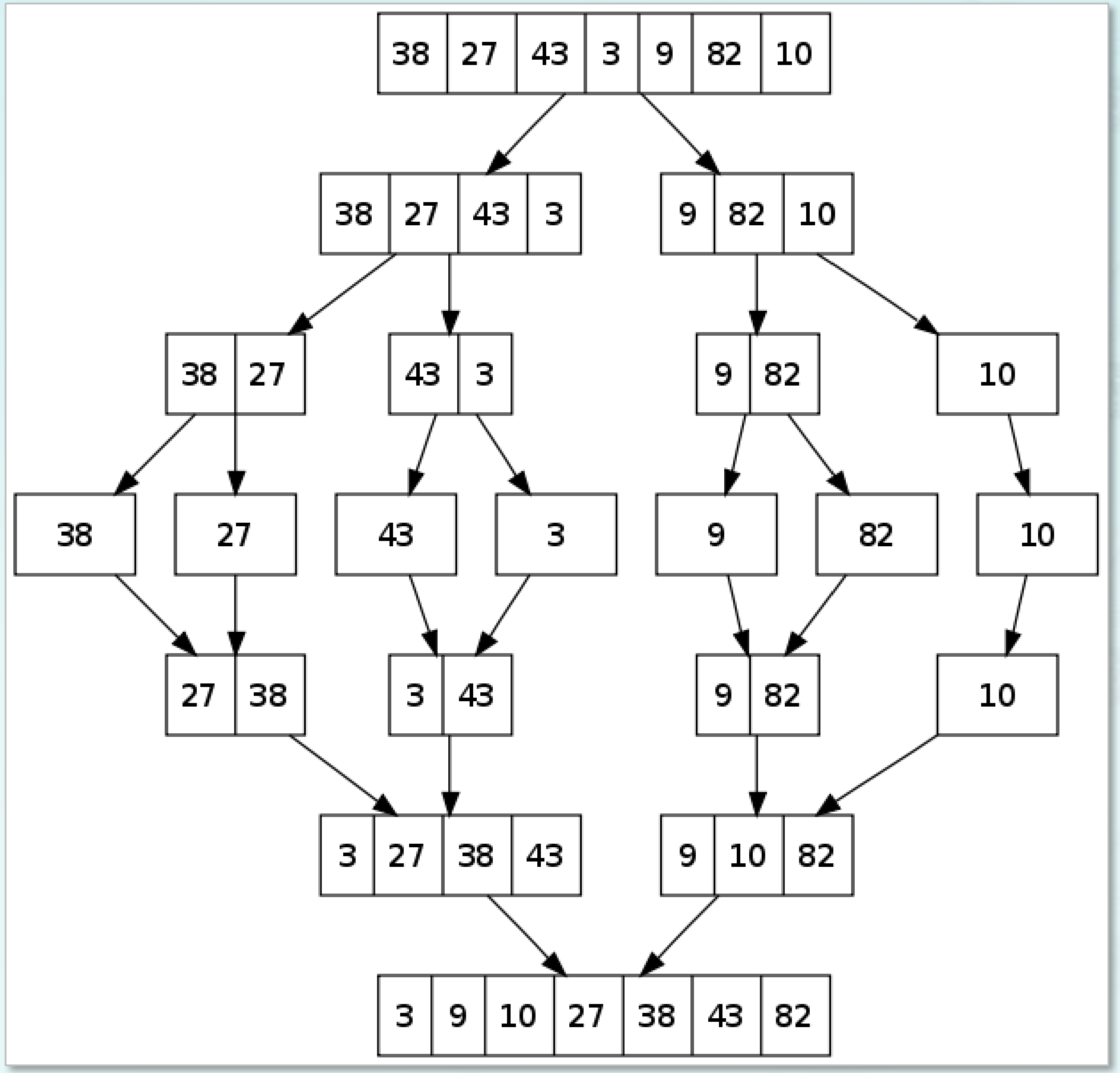
0	1	2	3	4	lo=5	6	mi=7	8	hi=9
2	5	8	9	16	23	31	56	62	71

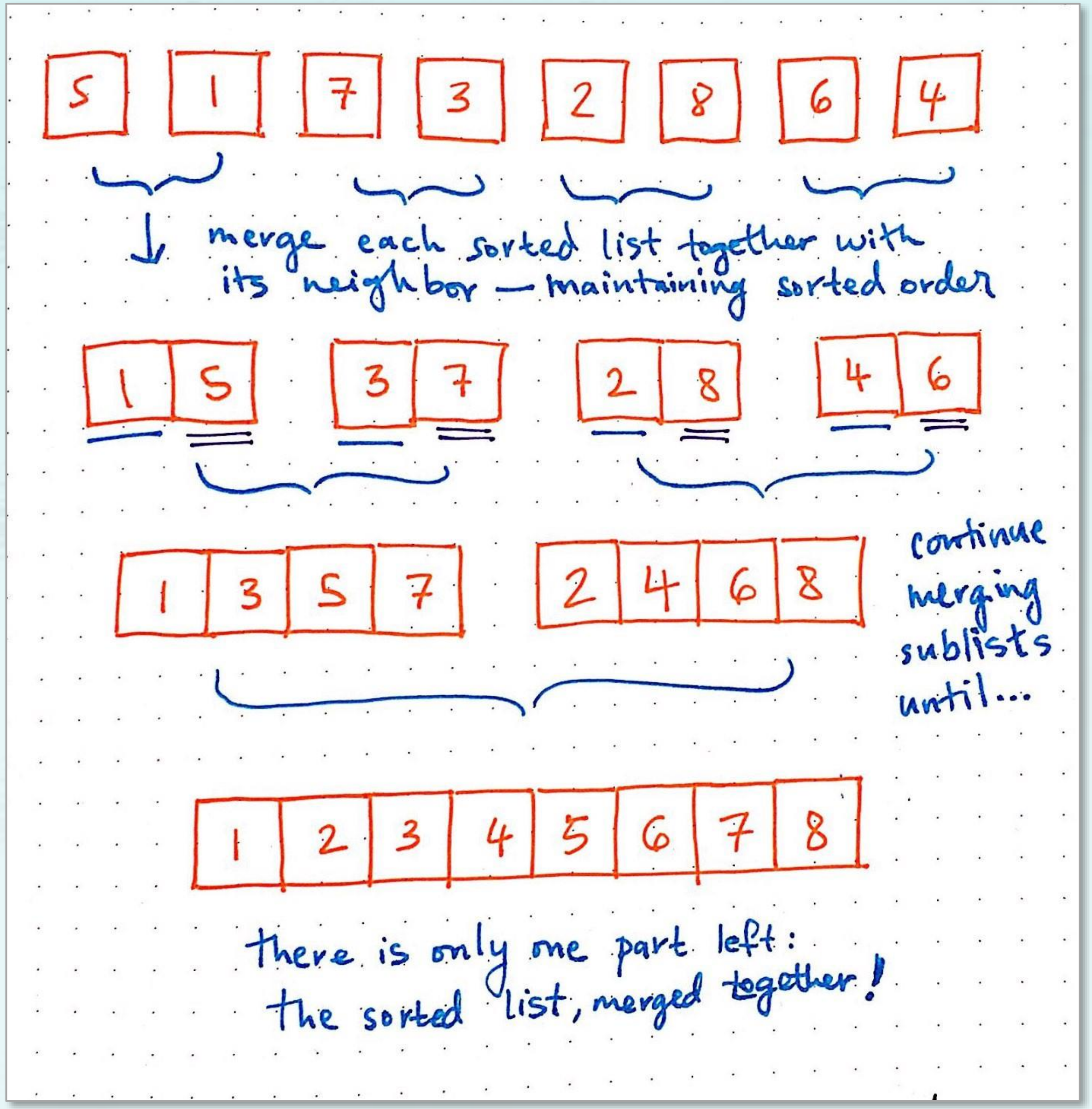
0	1	2	3	4	mi=5 lo=5	hi=6	7	8	9
2	5	8	9	16	23	31	56	62	71

```
int binarySearch(int list[], int key,
                             int lo, int hi) {
  if (lo > hi) return -1;

  mi = (lo + hi)/2;
  if (key == list[mi]) return mi;
  if (key <  list[mi])
    return binarySearch(list, key, lo, mi - 1);
  else
    return binarySearch(list, key, mi + 1, hi);
}
```

Mergesort: Coding

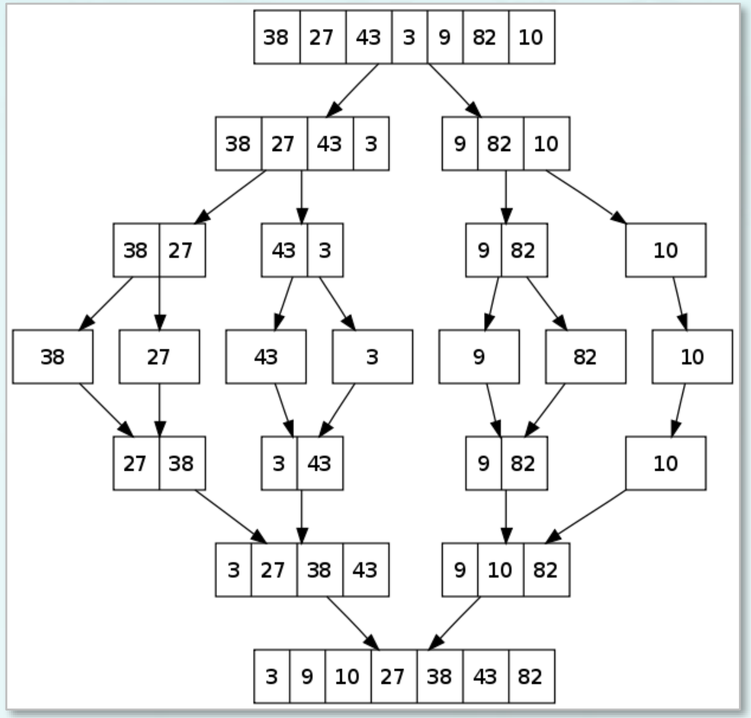




https://en.wikipedia.org/wiki/File:Merge_sort_algorithm_diagram.svg

Mergesort: Coding

```
mergeSort(a[], aux[], N, lo,  hi)
If hi > lo
  1. Find the middle to divide the array into two: mi = (lo+hi)/2
  2. Call mergeSort for 1st half: mergeSort(a, aux, N, lo,   mi)
  3. Call mergeSort for 2nd half: mergeSort(a, aux, N, mi+1, hi)
  4. Merge the two halves sorted: merge(a, aux, lo, mi, hi)
```



Mergesort: Coding

```
void mergeSort(char *a, char *aux, int N, int lo, int hi) {
  if (hi <= lo) return;
  int mi = lo + (hi - lo) / 2;                    // mi=(lo+hi)/2
  mergeSort (a, aux, N, lo,     mi);
  mergeSort (a, aux, N, mi + 1, hi);
  merge(a, aux, lo, mi, hi);
}

int main() {
  char a[]={'M','E','R','G','E','S','O','R','T','E','X','A','M','P','L','E'};
  cout << "UNSORTED: "; for (auto x: a) cout << x; cout << endl;
  int N = sizeof(a) / sizeof(a[0]);
  char *aux = new char[N];
  mergeSort(a, aux, N, 0, N - 1);
  cout << "  SORTED: "; for (auto x: a) cout << x; cout << endl;
}
```

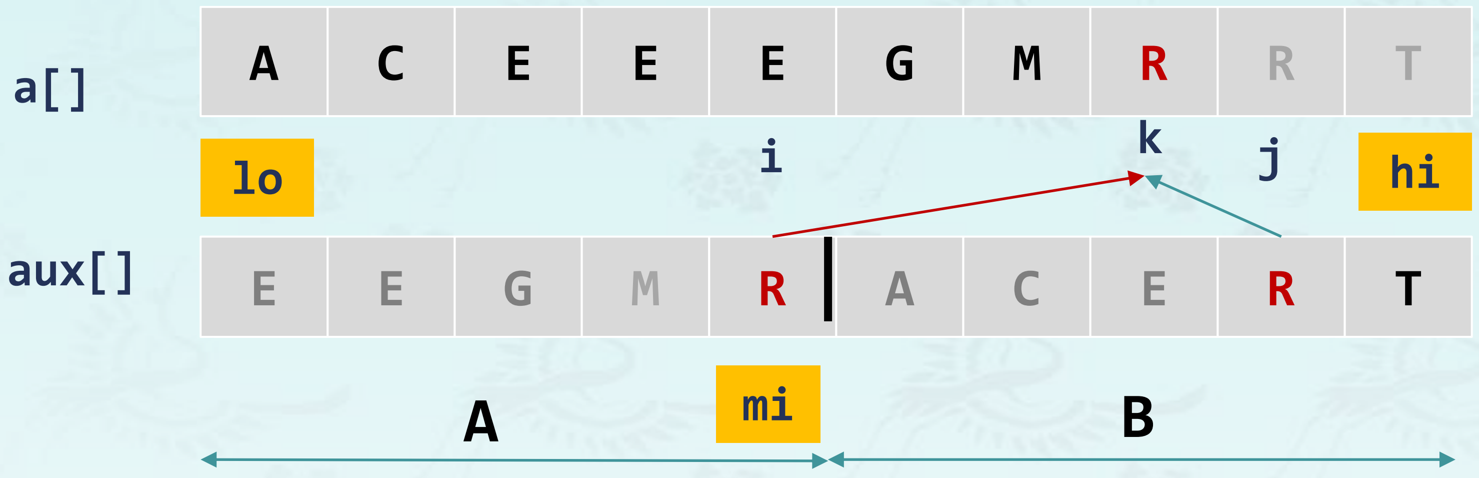
Mergesort: Coding

```
int isSorted(char *a, int i, int j){return a[i] <= a[j];}

void merge(char *a, char *aux, int lo, int mi, int hi) {
    assert(isSorted(a, lo,   mi));     // precondition: a[lo..mi]   sorted
    assert(isSorted(a, mi+1, hi));     // precondition: a[mi+1..hi] sorted
    for (int k = lo; k <= hi; k++) aux[k] = a[k];

    int i = lo;
    int j = mi + 1;
    for (int k = lo; k <= hi; k++) {
        if      (i > mi)           a[k] = aux[j++];     // A is exhausted, take B[j]
        else if (j > hi)           a[k] = aux[i++];     // B is exhausted, take A[i]
        else if (aux[j] < aux[i])  a[k] = aux[j++];     // B[j] <  A[i], take B[j]
        else                       a[k] = aux[i++];     // A[i] <= B[j], take A[i]
    }
    assert(isSorted(a, lo, hi));       // postcondition: a[lo..hi] sorted
}
```

Mergesort: Coding



Mergesort: Coding

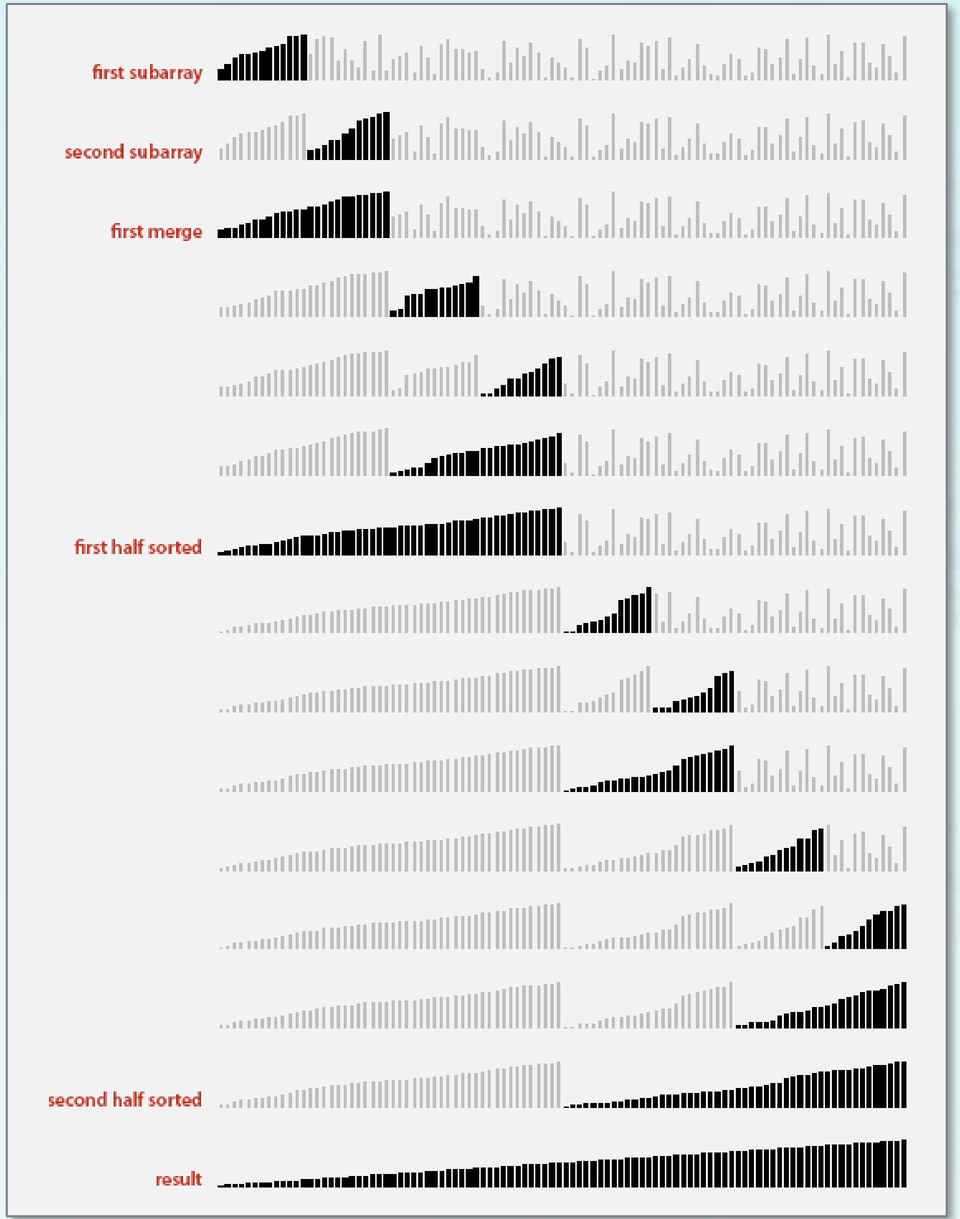
a[]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lo hi	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 1, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 4, 5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 6, 6, 7)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 5, 7)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
merge(a, aux, 0, 3, 7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
merge(a, aux, 8, 8, 9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
merge(a, aux, 10, 10, 11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
merge(a, aux, 8, 9, 11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, aux, 12, 12, 13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, aux, 14, 14, 15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
merge(a, aux, 12, 13, 15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, aux, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a, aux, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

result after recursive call

https://algs4.cs.princeton.edu/22mergeSort/

Mergesort: Coding



https://algs4.cs.princeton.edu/22mergeSort/

Assertion in C/C++

- **Assertion:** Statement to test assumptions about your program in Java.
  - Helps detect logic bugs.
  - Documents code.
- **Assert statement:** abort the program and print an error message (the function name and a line number) unless Boolean condition is true.

```
#include <cassert>
assert( isSorted(a, lo, hi) );
```

- **Can disable at runtime:** enabled by default.

```
#define DEBUG
g++ -DDEBUG
```

- **Best practices:** Use assertions to check internal invariants;
  - Assume assertions will be disabled in production code.
  - Do not use for external argument checking.

Mergesort: Quiz 1

1. Improvement by reducing the number of `merge()` function call.
   Some hints for this problem are provided in the following pages.
2. How many times did you spare `merge()` calls for "`MERGESORTEXAMPLE`" case?
   - Total number of **merge()** calls without your improvement: ______________
   - The number of **merge()** calls spared with your improvement: ________________
3. Identify those sets of char array groups that `merge()` call was unnecessary.

Mergesort: Quiz 1

- **Hint:** Do not invoke "merge()" function **if two halves are already sorted..**
  - Is the biggest item in first half ≤ the smallest item in second half?
  - For example, the following case should not call merge() since `J <= M`.

A	B	C	D	E	F	G	H	I	J	M	N	O	P	Q	R	S	T	U	V
A	B	C	D	E	F	G	H	I	J	M	N	O	P	Q	R	S	T	U	V

정렬들에 관한 좋은 자료를 읽어 보길 적극 추천합니다.
영어: https://medium.com/basecs/making-sense-of-merge-sort-part-1-49649a143478
한글: https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html

Mergesort: Quiz

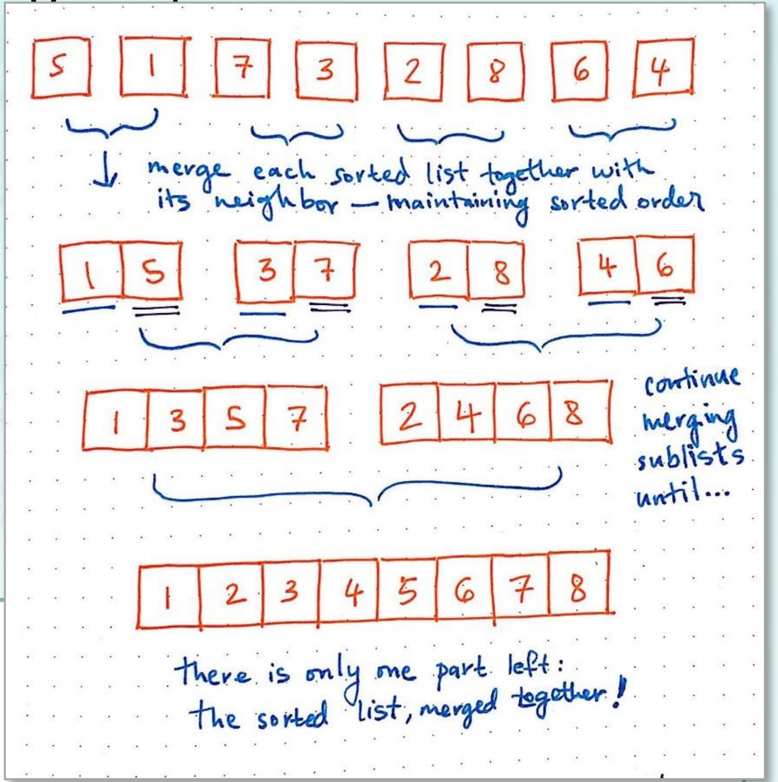
- **Hint:** Do not invoke "merge()" function **if two halves are already sorted..**
  - Is the biggest item in first half ≤ the smallest item in second half?
  - For example, the following case should not call merge() since `J <= M`.

A	B	C	D	E	F	G	H	I	J	M	N	O	P	Q	R	S	T	U	V
A	B	C	D	E	F	G	H	I	J	M	N	O	P	Q	R	S	T	U	V

```
void mergeSort(char *a, char *aux, int N, int lo, int hi) {
    if (hi <= lo)  return;
    int mi = lo + (hi - lo) / 2;
    mergeSort(a, aux, N, lo,     mi);
    mergeSort(a, aux, N, mi + 1, hi);
    if ( a[mi + 1] > a[mi] ) return;          // already sorted
    merge(a, aux, lo, mi, hi);
}
```

Mergesort: Quiz 2

- In the figure, which elements are compared in isSorted() at postcondition?
- Why isSorted() checks only two elements?
  Is this enough?




```
int isSorted(int *a, int i, int j){return a[i] <= a[j];}

void merge(int *a, char *aux, int lo, int mi, int hi) {
    assert(isSorted(a, lo,   mi));    // precondition: a[lo..mi]   sorted
    assert(isSorted(a, mi+1, hi));    // precondition: a[mi+1..hi] sorted
    for (int k = lo; k <= hi; k++) aux[k] = a[k];
     ......
    assert(isSorted(a, lo, hi));      // postcondition: a[lo..hi] sorted
}
```

Data Structures
Chapter 1

1. **Recursion**
   - Recursion
   - **Mergesort**
2. Performance Analysis
3. Asymptotic Analysis