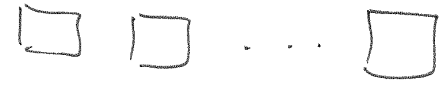
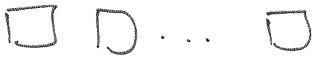
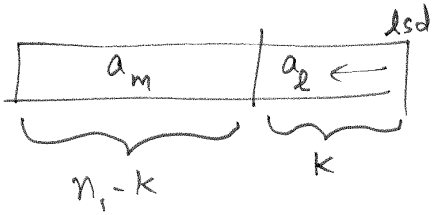
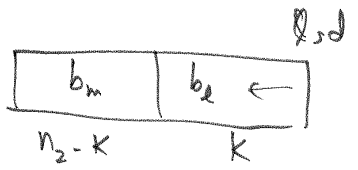
DAC algorithm for multiplication (LP1)

a  n_1 elements. $n_1 \geq n_2$

b  n_2 elements

Let $k = \lfloor \frac{n_2}{2} \rfloor$

a 



$$a * b = a_m * b_m * Base^{2k} + \left((a_m + a_\ell) * (b_m + b_\ell) - a_m * b_m - a_\ell * b_\ell \right) * Base^{k} + a_\ell * b_\ell$$

Sorting Input: Arrays	Extra Space	Time	
Merge Sort	$O(n)$	$O(n \log n)$	- Best sorting algorithm for internal/external applications
Heap Sort	$O(1)$ In-Place	$O(n \log n)$	- It is known that HeapSort is slower than Merge Sort.
QuickSort	$O(\log n)$ In-place	$O(n \log n)$ expected	- Good implementations of Q Sort are usually better than Merge Sort. (except for large values of n).

Partition1 (A, p, r)

Select i uniformly at random from [p..r]
and exchange A[i] ↔ A[r]

x ← A[r] // Pivot element

i ← p-1

// Loop invariant: A[p..i] ≤ x
// A[i+1..j-1] > x
// A[j..r-1] unprocessed
// A[r] = x

for j ← p to r-1 do
  if A[j] ≤ x then
    i ← i+1
    Exchange A[i] ↔ A[j]

// Bring pivot back to the middle
Exchange A[i+1] ↔ A[r]
// A[p..i] ≤ x , A[i+1] = x , A[i+2..r] > x
Return i+1

Partition2 (A, p, r) // Hoare's algorithm

Let x be chosen from A[p..r] uniformly at random

i ← p-1 j ← r+1

// LI: A[p..i] ≤ x , A[j..r] ≥ x

while true do
  do { i++ } while A[i] < x
  do { j-- } while A[j] > x
  if i ≥ j then
    return j
  Exchange A[i] ↔ A[j]

QuickSort (A, p, r) // Sort A[p..r]

if p < r then
  q ← Partition (A, p, r)
  QuickSort (A, p, q-1)
  QuickSort (A, q+1, r)

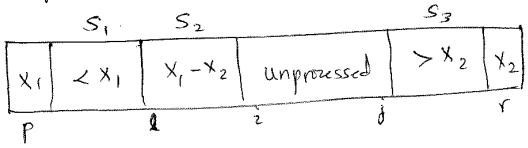
(q+1: if using Partition2)

Dual-Pivot Partition (Yaroslavskiy)

Choose 2 elements of A[p..r] at random, move them to the ends of A:

A[p] = x1 , A[r] = x2 , x1 ≤ x2

Loop invariant:



Unprocessed elements are processed from both ends:

Case 1: x1 ≤ A[i] ≤ x2
  S2 grows by 1 element ; i++

Case 2: A[i] < x1
  A[i] ↔ A[l] ; l++
  Exchange A[i] and left most element of S2 ; i++
  S1 grows by 1 element,
  S2 is shifted by 1 (effectively)

Case 3: A[j] > x2
  S3 grows by 1 element ; j--

Case 4: A[i] > x2 A[j] < x1

Circular swap: A[l] → A[i] → A[j] → A[l]

l++
i++
j--

Case 5: A[i] > x2 x1 ≤ A[j] ≤ x2

A[i] ↔ A[j]
i++
j--

At the end of the algorithm:

Swap A[p] ↔ A[l-1]
  A[j+1] ↔ A[r]

QuickSort:
  Dual-Pivot Partition
  Sort S1
  if x1 ≠ x2 Sort S2
  Sort S3