

Divide and Conquer

Text Book References:

Fundamentals of Computer Algorithms – Sartaj Sahni

Introduction to Algorithms – Thomas H Cormen

Unit II

Divide and conquer: The general method, Iterative and Divide and conquer for Binary search, Merge sort, Quick sort, Masters' theorem.

Divide-and-Conquer

Divide the problem into a number of sub-problems

- Similar sub-problems of smaller size

Conquer the sub-problems

- Solve the sub-problems recursively
- Sub-problem size small enough \Rightarrow solve the problems in straightforward manner

Combine the solutions of the sub-problems

- Obtain the solution for the original problem

Divide and Conquer Strategy:

- 1) Divide: Breaking the problem into sub problems that are themselves smaller instances of the same type of problem.
- 2) Recursion: Recursively solving these sub problems.
- 3) Conquer: Appropriately combining their answers.

Control abstraction:

DAndC(P)

{

if (Small(P))

// P is very small so that a solution
// is obvious

return solution(n);

else

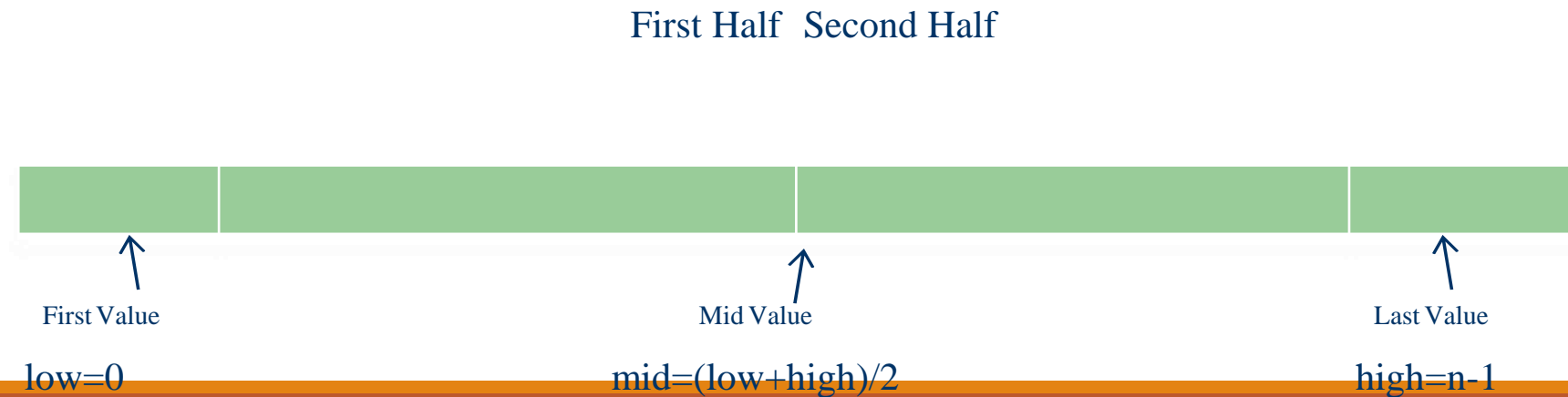
divide the problem P into k
sub problems P_1, P_2, \dots, P_k

return (combine(DAndC(P_1), DAndC(P_2),
... DAndC(P_k)))

}

Binary Search

- This technique works better on sorted arrays and can be applied only on sorted arrays.
- Not applied on Linked Lists.
- Requires less number of comparisons than linear search.
- Efficiency: $O(\log_2 n)$.
- Logic behind the technique:



Binary Search

Iterative

BinarySearch(A, n, x)

```
{
  low ← 0
  high ← n-1
  while (low ≤ high)
  {
    mid ← (low + high) / 2
    if (x == A[mid])
      return mid
    else if (x < A[mid])
      high ← mid - 1
    else
      low ← mid + 1
  }
  return -1
}
```

$O(\log_2 n)$

Recursive

BinarySearch(A, low, high, x)

```
{
  mid ← (low + high) / 2
  if (low > high)
    return -1
  mid ← (low + (high)) / 2
  if (x == A[mid])
    return mid
  else if x < A[mid]
    return BinarySearch(A, low, mid-1, x)
  else
    return BinarySearch(A, mid+1, high, x)
}
```

$$T(n) = T(n/2) + c$$

$$T(1) = c$$

$$\Theta(\log_2 n)$$

n	c
n/2	c
n/4	c
...	...
2	c
1	c
0	c

Binary Search

Recursive

BinarySearch(A, low, high, x) — $T(n)$

```
if (low > high) // if (low == high)
    return -1      // if (A[low] == key)
                    // return low;
else
    mid ← (low + high) / 2
    if (x == A[mid])
        return mid
    else if (x < A[mid])
        return BinarySearch(A, low, mid-1, x)
    else
        return BinarySearch(A, mid+1, high, x)
```

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + 1 & n > 1 \end{cases}$$

$\Rightarrow \theta(\log n)$

By using Master's theorem
 $a = 1, b = 2, k = 0, p = 0$
 $b^k = 2^0 = 1$
Case 2: $p > -1$
 $T(n) = \theta(n^{\log_b a} \log^p n)$
 $= \theta(n^{\log_2 1} \log^0 n)$
 $= \theta(n^0 \log n)$
 $= \theta(\log n)$

Recursion tree diagram:

```
graph TD
    Tn["T(n)"] --> Tn2["T(n/2)"]
    Tn --> Tn2
    Tn2 --> Tn4["T(n/4)"]
    Tn2 --> Tn4
    Tn4 --> Tn8["T(n/8)"]
    Tn4 --> Tn8
    Tn8 --> Tn16["T(n/16)"]
    Tn8 --> Tn16
    Tn16 --> Tn32["T(n/32)"]
    Tn16 --> Tn32
    Tn32 --> Tn64["T(n/64)"]
    Tn32 --> Tn64
    Tn64 --> Tn128["T(n/128)"]
    Tn64 --> Tn128
    Tn128 --> Tn256["T(n/256)"]
    Tn128 --> Tn256
    Tn256 --> Tn512["T(n/512)"]
    Tn256 --> Tn512
    Tn512 --> Tn1024["T(n/1024)"]
    Tn512 --> Tn1024
    Tn1024 --> Tn2048["T(n/2048)"]
    Tn1024 --> Tn2048
    Tn2048 --> Tn4096["T(n/4096)"]
    Tn2048 --> Tn4096
    Tn4096 --> Tn8192["T(n/8192)"]
    Tn4096 --> Tn8192
    Tn8192 --> Tn16384["T(n/16384)"]
    Tn8192 --> Tn16384
    Tn16384 --> Tn32768["T(n/32768)"]
    Tn16384 --> Tn32768
    Tn32768 --> Tn65536["T(n/65536)"]
    Tn32768 --> Tn65536
    Tn65536 --> Tn131072["T(n/131072)"]
    Tn65536 --> Tn131072
    Tn131072 --> Tn262144["T(n/262144)"]
    Tn131072 --> Tn262144
    Tn262144 --> Tn524288["T(n/524288)"]
    Tn262144 --> Tn524288
    Tn524288 --> Tn1048576["T(n/1048576)"]
    Tn524288 --> Tn1048576
    Tn1048576 --> Tn2097152["T(n/2097152)"]
    Tn1048576 --> Tn2097152
    Tn2097152 --> Tn4194304["T(n/4194304)"]
    Tn2097152 --> Tn4194304
    Tn4194304 --> Tn8388608["T(n/8388608)"]
    Tn4194304 --> Tn8388608
    Tn8388608 --> Tn16777216["T(n/16777216)"]
    Tn8388608 --> Tn16777216
    Tn16777216 --> Tn33554432["T(n/33554432)"]
    Tn16777216 --> Tn33554432
    Tn33554432 --> Tn67108864["T(n/67108864)"]
    Tn33554432 --> Tn67108864
    Tn67108864 --> Tn134217728["T(n/134217728)"]
    Tn67108864 --> Tn134217728
    Tn134217728 --> Tn268435456["T(n/268435456)"]
    Tn134217728 --> Tn268435456
    Tn268435456 --> Tn536870912["T(n/536870912)"]
    Tn268435456 --> Tn536870912
    Tn536870912 --> Tn1073741824["T(n/1073741824)"]
    Tn536870912 --> Tn1073741824
    Tn1073741824 --> Tn2147483648["T(n/2147483648)"]
    Tn1073741824 --> Tn2147483648
    Tn2147483648 --> Tn4294967296["T(n/4294967296)"]
    Tn2147483648 --> Tn4294967296
    Tn4294967296 --> Tn8589934592["T(n/8589934592)"]
    Tn4294967296 --> Tn8589934592
    Tn8589934592 --> Tn17179869184["T(n/17179869184)"]
    Tn8589934592 --> Tn17179869184
    Tn17179869184 --> Tn34359738368["T(n/34359738368)"]
    Tn17179869184 --> Tn34359738368
    Tn34359738368 --> Tn68719476736["T(n/68719476736)"]
    Tn34359738368 --> Tn68719476736
    Tn68719476736 --> Tn137438953472["T(n/137438953472)"]
    Tn68719476736 --> Tn137438953472
    Tn137438953472 --> Tn274877906944["T(n/274877906944)"]
    Tn137438953472 --> Tn274877906944
    Tn274877906944 --> Tn549755813888["T(n/549755813888)"]
    Tn274877906944 --> Tn549755813888
    Tn549755813888 --> Tn1099511627776["T(n/1099511627776)"]
    Tn549755813888 --> Tn1099511627776
    Tn1099511627776 --> Tn2199023255552["T(n/2199023255552)"]
    Tn1099511627776 --> Tn2199023255552
    Tn2199023255552 --> Tn4398046511104["T(n/4398046511104)"]
    Tn2199023255552 --> Tn4398046511104
    Tn4398046511104 --> Tn8796093022208["T(n/8796093022208)"]
    Tn4398046511104 --> Tn8796093022208
    Tn8796093022208 --> Tn17592186044416["T(n/17592186044416)"]
    Tn8796093022208 --> Tn17592186044416
    Tn17592186044416 --> Tn35184372088832["T(n/35184372088832)"]
    Tn17592186044416 --> Tn35184372088832
    Tn35184372088832 --> Tn70368744177664["T(n/70368744177664)"]
    Tn35184372088832 --> Tn70368744177664
    Tn70368744177664 --> Tn140737488355328["T(n/140737488355328)"]
    Tn70368744177664 --> Tn140737488355328
    Tn140737488355328 --> Tn281474976710656["T(n/281474976710656)"]
    Tn140737488355328 --> Tn281474976710656
    Tn281474976710656 --> Tn562949953421312["T(n/562949953421312)"]
    Tn281474976710656 --> Tn562949953421312
    Tn562949953421312 --> Tn1125899906842624["T(n/1125899906842624)"]
    Tn562949953421312 --> Tn1125899906842624
    Tn1125899906842624 --> Tn2251799813685248["T(n/2251799813685248)"]
    Tn1125899906842624 --> Tn2251799813685248
    Tn2251799813685248 --> Tn4503599627370496["T(n/4503599627370496)"]
    Tn2251799813685248 --> Tn4503599627370496
    Tn4503599627370496 --> Tn9007199254740992["T(n/9007199254740992)"]
    Tn4503599627370496 --> Tn9007199254740992
    Tn9007199254740992 --> Tn18014398509481984["T(n/18014398509481984)"]
    Tn9007199254740992 --> Tn18014398509481984
    Tn18014398509481984 --> Tn36028797018963968["T(n/36028797018963968)"]
    Tn18014398509481984 --> Tn36028797018963968
    Tn36028797018963968 --> Tn72057594037927936["T(n/72057594037927936)"]
    Tn36028797018963968 --> Tn72057594037927936
    Tn72057594037927936 --> Tn144115188075855872["T(n/144115188075855872)"]
    Tn72057594037927936 --> Tn144115188075855872
    Tn144115188075855872 --> Tn288230376151711744["T(n/288230376151711744)"]
    Tn144115188075855872 --> Tn288230376151711744
    Tn288230376151711744 --> Tn576460752303423488["T(n/576460752303423488)"]
    Tn288230376151711744 --> Tn576460752303423488
    Tn576460752303423488 --> Tn1152921504606846976["T(n/1152921504606846976)"]
    Tn576460752303423488 --> Tn1152921504606846976
    Tn1152921504606846976 --> Tn2305843009213693952["T(n/2305843009213693952)"]
    Tn1152921504606846976 --> Tn2305843009213693952
    Tn2305843009213693952 --> Tn4611686018427387904["T(n/4611686018427387904)"]
    Tn2305843009213693952 --> Tn4611686018427387904
    Tn4611686018427387904 --> Tn9223372036854775808["T(n/9223372036854775808)"]
    Tn4611686018427387904 --> Tn9223372036854775808
    Tn9223372036854775808 --> Tn18446744073709551616["T(n/18446744073709551616)"]
    Tn9223372036854775808 --> Tn18446744073709551616
    Tn18446744073709551616 --> Tn36893488147419103232["T(n/36893488147419103232)"]
    Tn18446744073709551616 --> Tn36893488147419103232
    Tn36893488147419103232 --> Tn73786976294838206464["T(n/73786976294838206464)"]
    Tn36893488147419103232 --> Tn73786976294838206464
    Tn73786976294838206464 --> Tn147573952589676412928["T(n/147573952589676412928)"]
    Tn73786976294838206464 --> Tn147573952589676412928
    Tn147573952589676412928 --> Tn295147905179352825856["T(n/295147905179352825856)"]
    Tn147573952589676412928 --> Tn295147905179352825856
    Tn295147905179352825856 --> Tn590295810358705651712["T(n/590295810358705651712)"]
    Tn295147905179352825856 --> Tn590295810358705651712
    Tn590295810358705651712 --> Tn1180591620717411303424["T(n/1180591620717411303424)"]
    Tn590295810358705651712 --> Tn1180591620717411303424
    Tn1180591620717411303424 --> Tn2361183241434822606848["T(n/2361183241434822606848)"]
    Tn1180591620717411303424 --> Tn2361183241434822606848
    Tn2361183241434822606848 --> Tn4722366482869645213696["T(n/4722366482869645213696)"]
    Tn2361183241434822606848 --> Tn4722366482869645213696
    Tn4722366482869645213696 --> Tn9444732965739290427392["T(n/9444732965739290427392)"]
    Tn4722366482869645213696 --> Tn9444732965739290427392
    Tn9444732965739290427392 --> Tn18889465931478580854784["T(n/18889465931478580854784)"]
    Tn9444732965739290427392 --> Tn18889465931478580854784
    Tn18889465931478580854784 --> Tn37778931862957161709568["T(n/37778931862957161709568)"]
    Tn18889465931478580854784 --> Tn37778931862957161709568
    Tn37778931862957161709568 --> Tn75557863725914323419136["T(n/75557863725914323419136)"]
    Tn37778931862957161709568 --> Tn75557863725914323419136
    Tn75557863725914323419136 --> Tn151115727451828646838272["T(n/151115727451828646838272)"]
    Tn75557863725914323419136 --> Tn151115727451828646838272
    Tn151115727451828646838272 --> Tn302231454903657293676544["T(n/302231454903657293676544)"]
    Tn151115727451828646838272 --> Tn302231454903657293676544
    Tn302231454903657293676544 --> Tn604462909807314587353088["T(n/604462909807314587353088)"]
    Tn302231454903657293676544 --> Tn604462909807314587353088
    Tn604462909807314587353088 --> Tn1208925819614629174706176["T(n/1208925819614629174706176)"]
    Tn604462909807314587353088 --> Tn1208925819614629174706176
    Tn1208925819614629174706176 --> Tn2417851639229258349412352["T(n/2417851639229258349412352)"]
    Tn1208925819614629174706176 --> Tn2417851639229258349412352
    Tn2417851639229258349412352 --> Tn4835703278458516698824704["T(n/4835703278458516698824704)"]
    Tn2417851639229258349412352 --> Tn4835703278458516698824704
    Tn4835703278458516698824704 --> Tn9671406556917033397649408["T(n/9671406556917033397649408)"]
    Tn4835703278458516698824704 --> Tn9671406556917033397649408
    Tn9671406556917033397649408 --> Tn19342813113834066795298816["T(n/19342813113834066795298816)"]
    Tn9671406556917033397649408 --> Tn19342813113834066795298816
    Tn19342813113834066795298816 --> Tn38685626227668133590597632["T(n/38685626227668133590597632)"]
    Tn19342813113834066795298816 --> Tn38685626227668133590597632
    Tn38685626227668133590597632 --> Tn77371252455336267181195264["T(n/77371252455336267181195264)"]
    Tn38685626227668133590597632 --> Tn77371252455336267181195264
    Tn77371252455336267181195264 --> Tn154742504910672534362390528["T(n/154742504910672534362390528)"]
    Tn77371252455336267181195264 --> Tn154742504910672534362390528
    Tn154742504910672534362390528 --> Tn309485009821345068724781056["T(n/309485009821345068724781056)"]
    Tn154742504910672534362390528 --> Tn309485009821345068724781056
    Tn309485009821345068724781056 --> Tn618970019642690137449562112["T(n/618970019642690137449562112)"]
    Tn309485009821345068724781056 --> Tn618970019642690137449562112
    Tn618970019642690137449562112 --> Tn1237940039285380274899124224["T(n/1237940039285380274899124224)"]
    Tn618970019642690137449562112 --> Tn1237940039285380274899124224
    Tn1237940039285380274899124224 --> Tn2475880078570760549798248448["T(n/2475880078570760549798248448)"]
    Tn1237940039285380274899124224 --> Tn2475880078570760549798248448
    Tn2475880078570760549798248448 --> Tn4951760157141521099596496896["T(n/4951760157141521099596496896)"]
    Tn2475880078570760549798248448 --> Tn4951760157141521099596496896
    Tn4951760157141521099596496896 --> Tn9903520314283042199192993792["T(n/9903520314283042199192993792)"]
    Tn4951760157141521099596496896 --> Tn9903520314283042199192993792
    Tn9903520314283042199192993792 --> Tn19807040628566084398385987584["T(n/19807040628566084398385987584)"]
    Tn9903520314283042199192993792 --> Tn19807040628566084398385987584
    Tn19807040628566084398385987584 --> Tn39614081257132168796771975168["T(n/39614081257132168796771975168)"]
    Tn19807040628566084398385987584 --> Tn39614081257132168796771975168
    Tn39614081257132168796771975168 --> Tn79228162514264337593543950336["T(n/79228162514264337593543950336)"]
    Tn39614081257132168796771975168 --> Tn79228162514264337593543950336
    Tn79228162514264337593543950336 --> Tn158456325028528675187087900672["T(n/158456325028528675187087900672)"]
    Tn79228162514264337593543950336 --> Tn158456325028528675187087900672
    Tn158456325028528675187087900672 --> Tn316912650057057350374175801344["T(n/316912650057057350374175801344)"]
    Tn158456325028528675187087900672 --> Tn316912650057057350374175801344
    Tn316912650057057350374175801344 --> Tn633825300114114700748351602688["T(n/633825300114114700748351602688)"]
    Tn316912650057057350374175801344 --> Tn633825300114114700748351602688
    Tn633825300114114700748351602688 --> Tn1267650600228229401496703205376["T(n/1267650600228229401496703205376)"]
    Tn633825300114114700748351602688 --> Tn1267650600228229401496703205376
    Tn1267650600228229401496703205376 --> Tn2535301200456458802993406410752["T(n/2535301200456458802993406410752)"]
    Tn1267650600228229401496703205376 --> Tn2535301200456458802993406410752
    Tn2535301200456458802993406410752 --> Tn5070602400912917605986812821504["T(n/5070602400912917605986812821504)"]
    Tn2535301200456458802993406410752 --> Tn5070602400912917605986812821504
    Tn5070602400912917605986812821504 --> Tn10141204801825835211973625643008["T(n/10141204801825835211973625643008)"]
    Tn5070602400912917605986812821504 --> Tn10141204801825835211973625643008
    Tn10141204801825835211973625643008 --> Tn20282409603651670423947251286016["T(n/20282409603651670423947251286016)"]
    Tn10141204801825835211973625643008 --> Tn20282409603651670423947251286016
    Tn20282409603651670423947251286016 --> Tn40564819207303340847894502572032["T(n/40564819207303340847894502572032)"]
    Tn20282409603651670423947251286016 --> Tn40564819207303340847894502572032
    Tn40564819207303340847894502572032 --> Tn81129638414606681695789005144064["T(n/81129638414606681695789005144064)"]
    Tn40564819207303340847894502572032 --> Tn81129638414606681695789005144064
    Tn81129638414606681695789005144064 --> Tn162259276829213363391578010288128["T(n/162259276829213363391578010288128)"]
    Tn81129638414606681695789005144064 --> Tn162259276829213363391578010288128
    Tn162259276829213363391578010288128 --> Tn324518553658426726783156020576256["T(n/324518553658426726783156020576256)"]
    Tn162259276829213363391578010288128 --> Tn324518553658426726783156020576256
    Tn324518553
```

Merge Sort Approach

To sort an array $A[p \dots r]$:

Divide

- Divide the n -element sequence to be sorted into two subsequences of $n/2$ elements each

Conquer

- Sort the subsequences recursively using merge sort
- When the size of the sequences is 1 there is nothing more to do

Combine

- Merge the two sorted subsequences

Merge Sort

Alg.: MERGE-SORT(A, p, r)

if $p < r$

 then $q \leftarrow \lfloor (p + r)/2 \rfloor$

 MERGE-SORT(A, p, q)

 MERGE-SORT($A, q + 1, r$)

 MERGE(A, p, q, r)

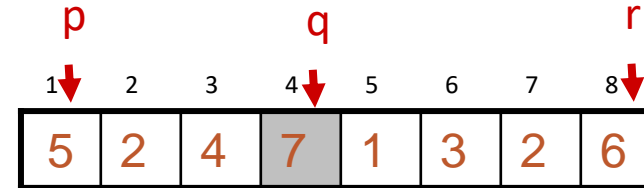
Check for base case

Divide

Conquer

Conquer

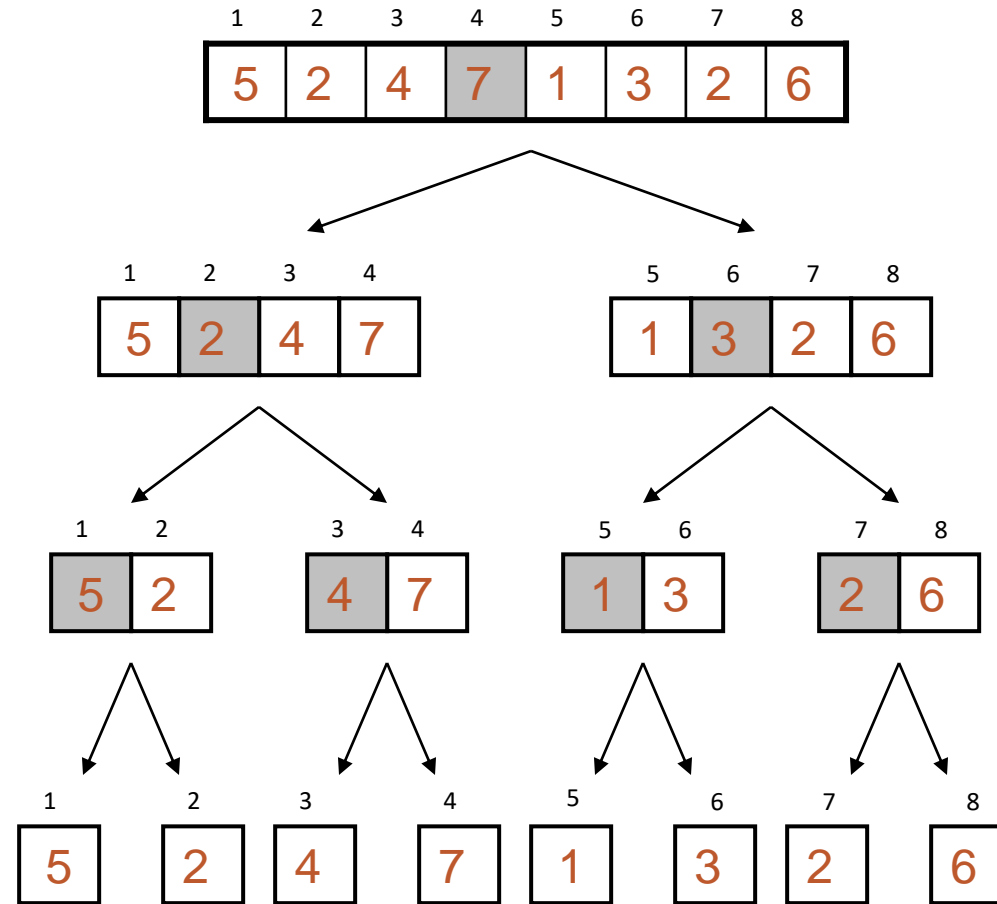
Combine



Initial call: MERGE-SORT($A, 1, n$)

Example – n Power of 2

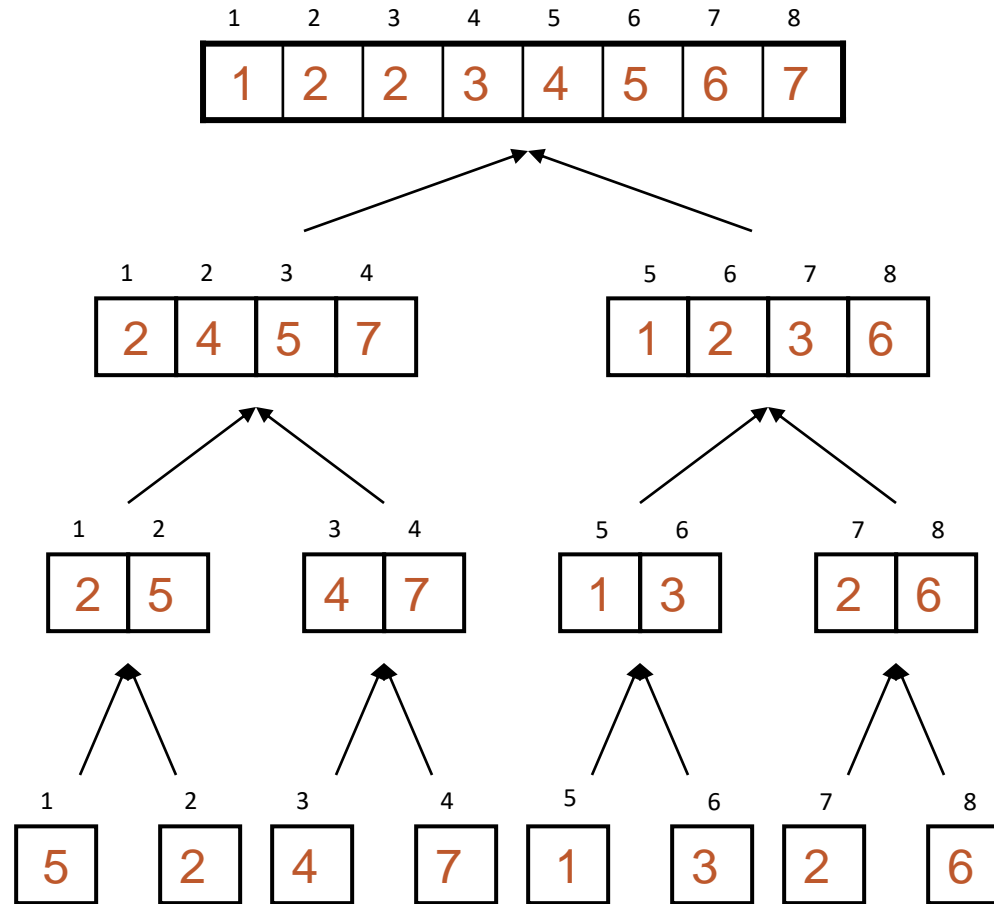
Divide



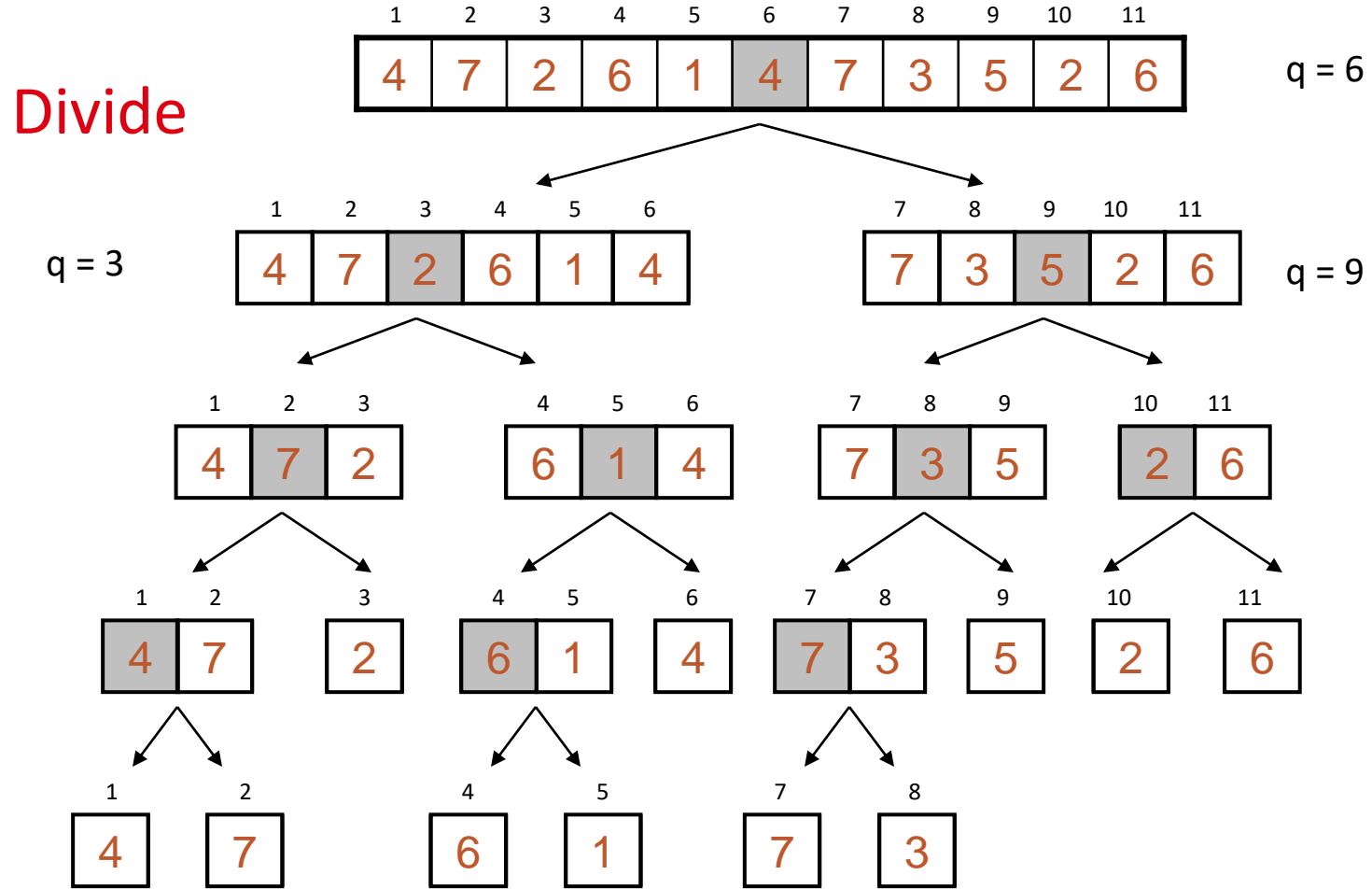
$q = 4$

Example – n Power of 2

Conquer
and
Merge

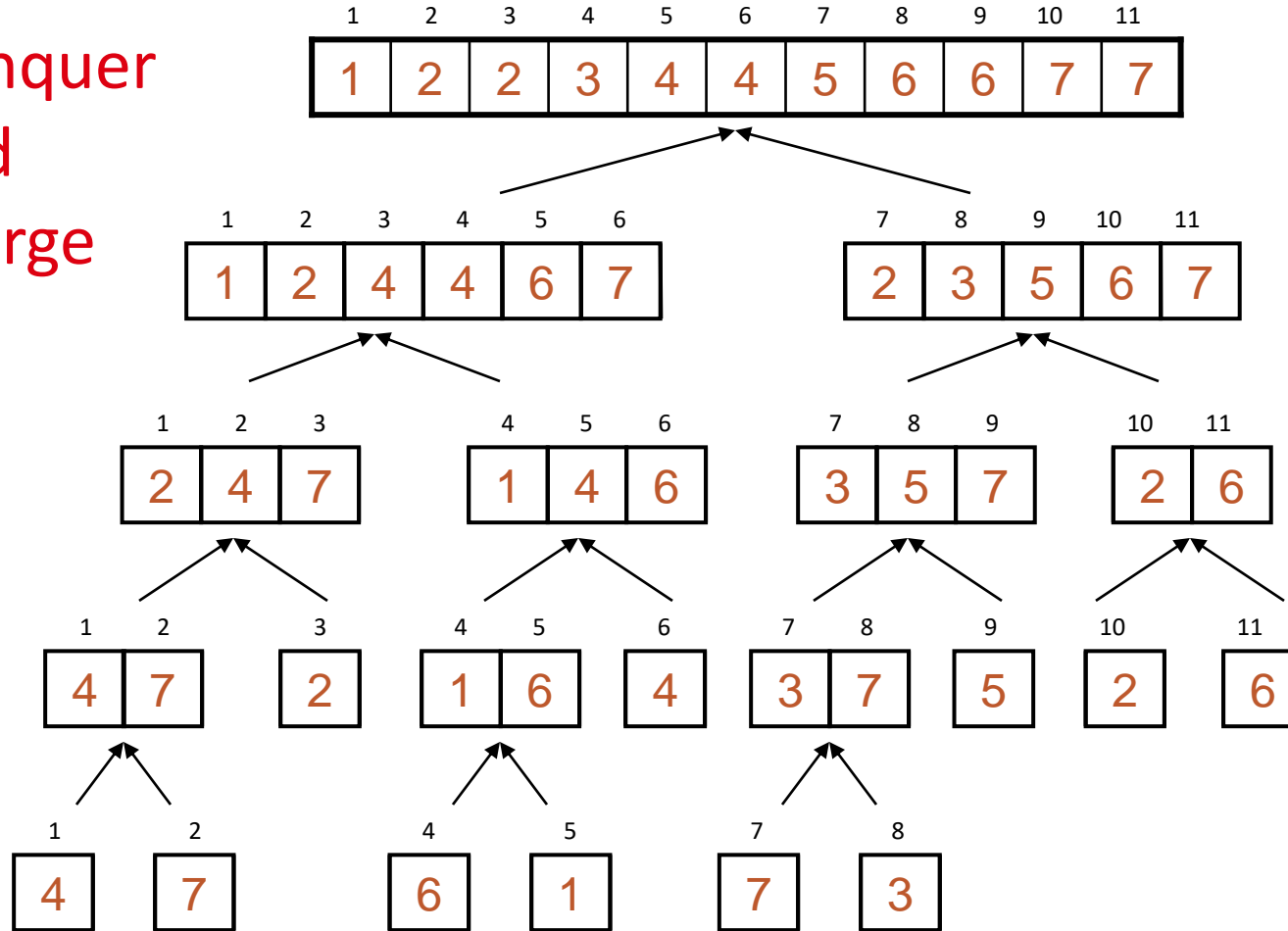


Example – n Not a Power of 2

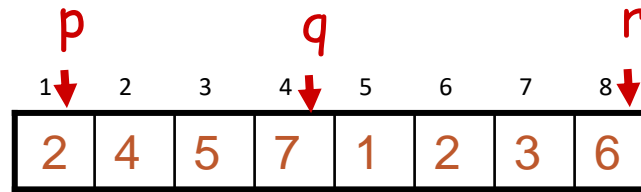


Example – n Not a Power of 2

Conquer
and
Merge



Merging



Input: Array A and indices p, q, r such that $p \leq q < r$

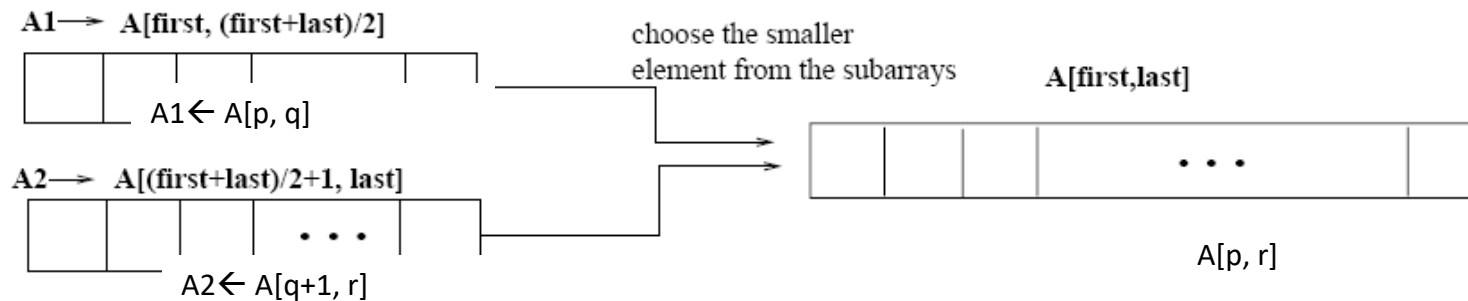
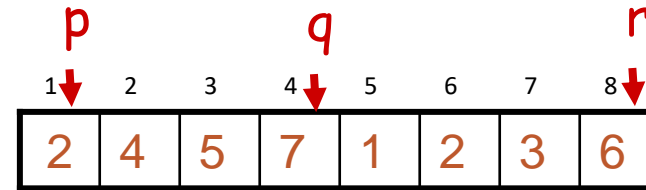
- Subarrays $A[p \dots q]$ and $A[q + 1 \dots r]$ are sorted

Output: One single sorted subarray $A[p \dots r]$

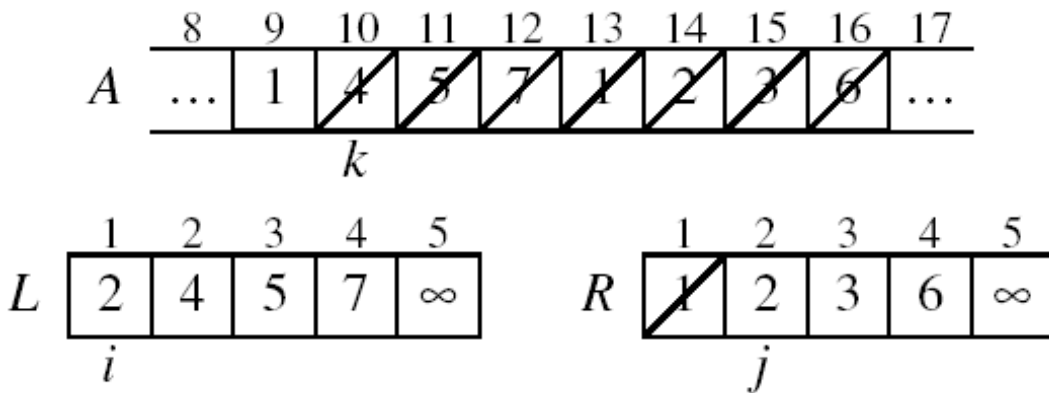
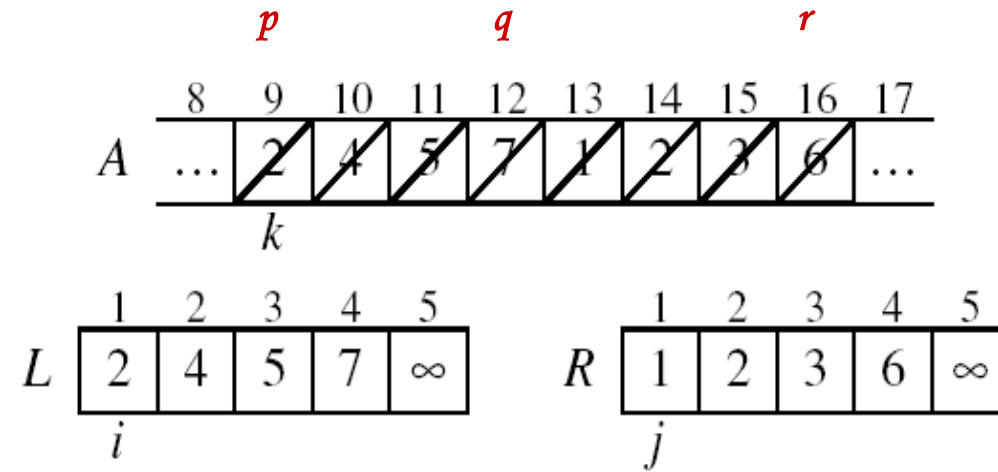
Merging

Idea for merging:

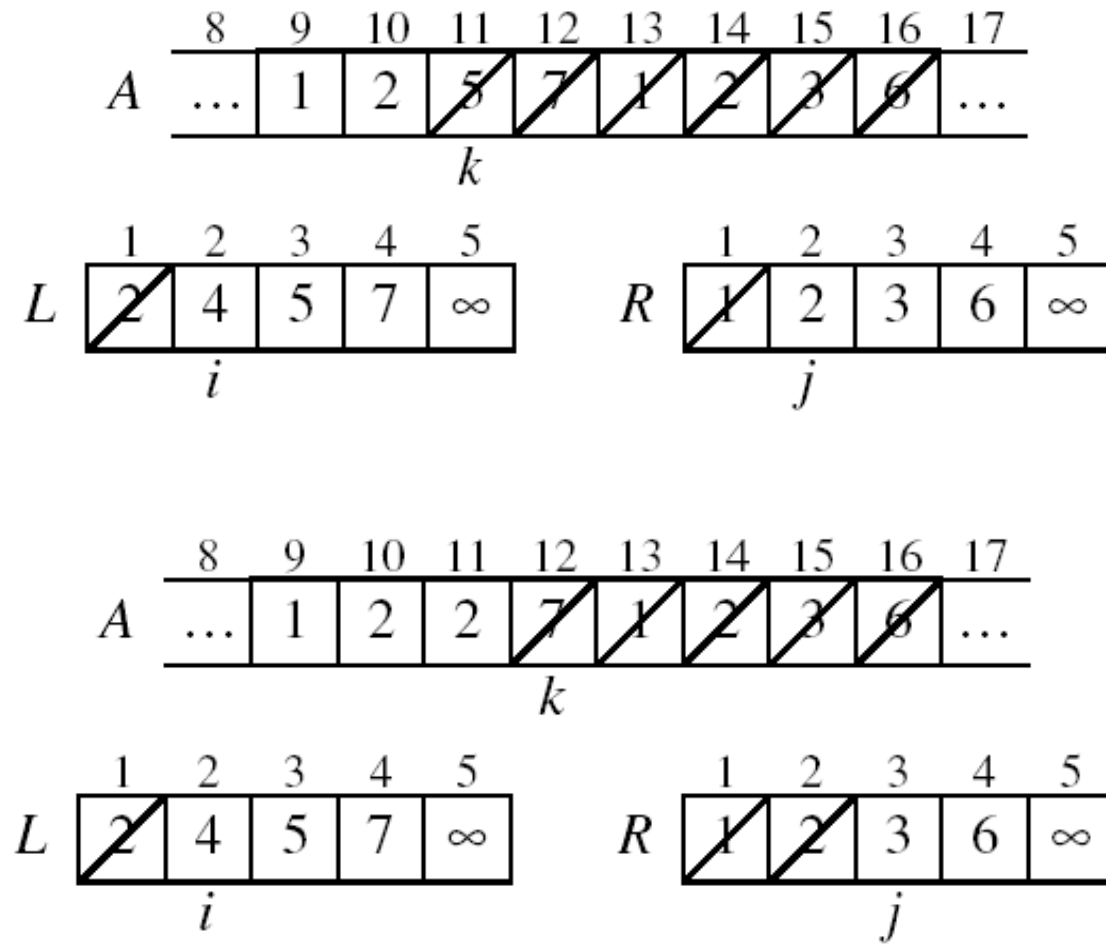
- Two piles of sorted cards
- Choose the smaller of the two top cards
- Remove it and place it in the output pile
- Repeat the process until one pile is empty
- Take the remaining input pile and place it face-down onto the output pile



Example: MERGE(A, 9, 12, 16)



Example: MERGE(A, 9, 12, 16)



Example (cont.)

	8	9	10	11	12	13	14	15	16	17
A	...	1	2	2	3	1	2	3	6	...

k

	1	2	3	4	5
L	2	4	5	7	∞

i

	1	2	3	4	5
R	1	2	3	6	∞

j

	8	9	10	11	12	13	14	15	16	17
A	...	1	2	2	3	4	2	3	6	...

k

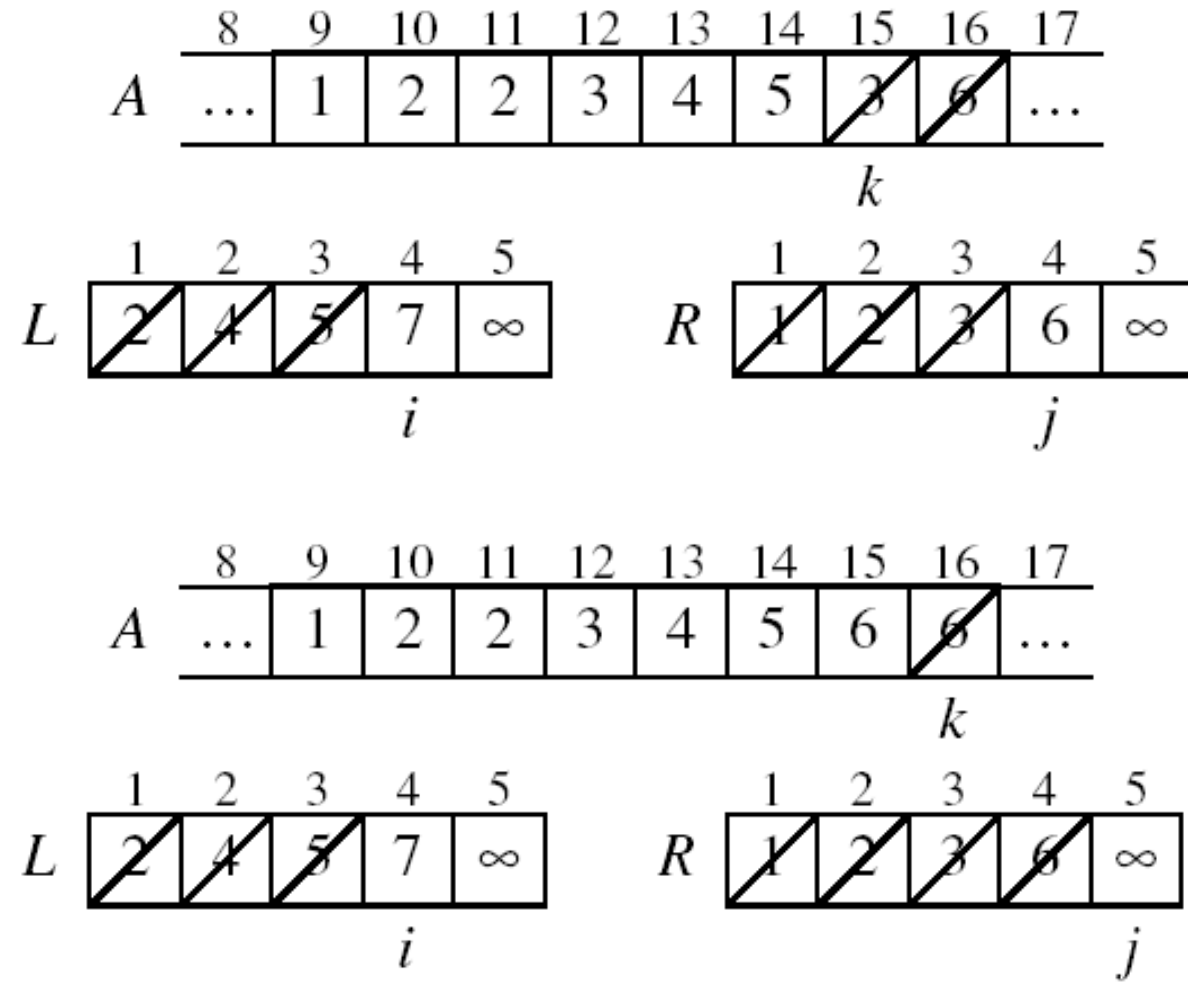
	1	2	3	4	5
L	2	4	5	7	∞

i

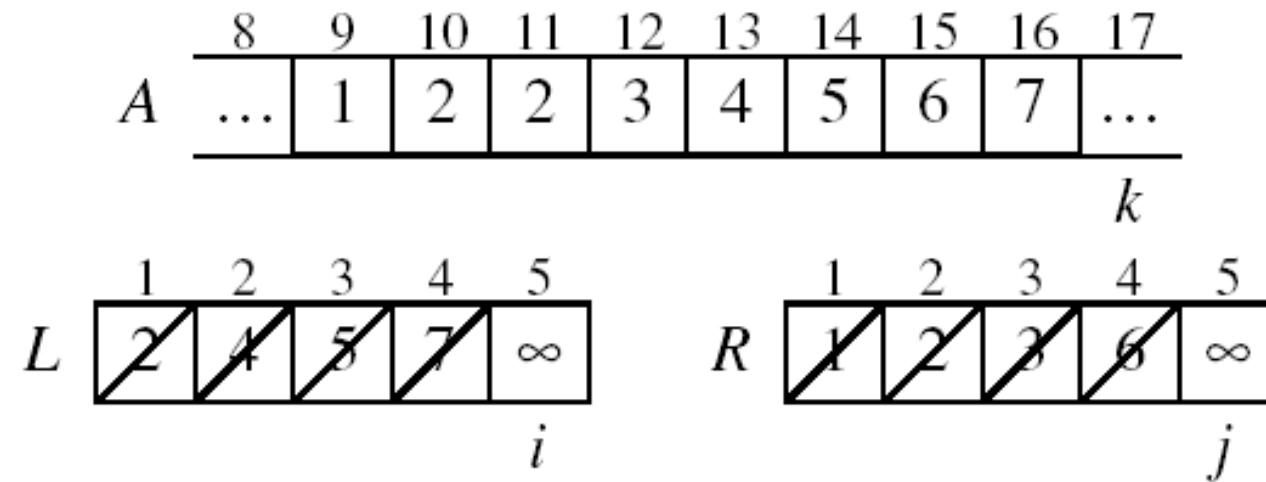
	1	2	3	4	5
R	1	2	3	6	∞

j

Example (cont.)



Example (cont.)

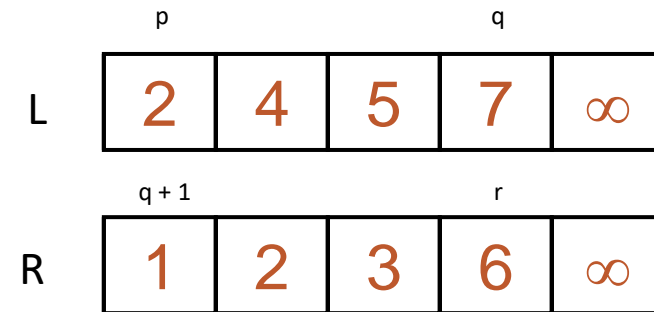
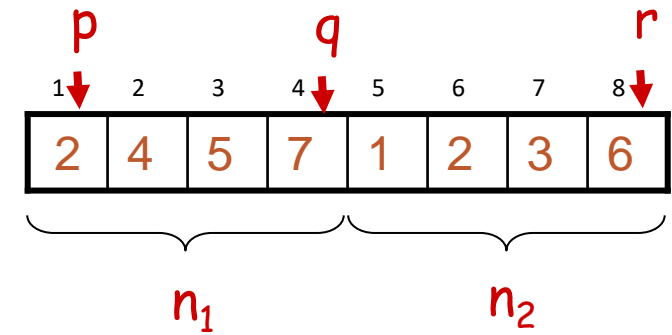


Done!

Merge - Pseudocode

Alg.: MERGE(A, p, q, r)

1. Compute n_1 and n_2
2. Copy the first n_1 elements into
3. $L[1 \dots n_1 + 1]$ and the next n_2 elements into $R[1 \dots n_2 + 1]$
4. $L[n_1 + 1] \leftarrow \infty$; $R[n_2 + 1] \leftarrow \infty$
5. $i \leftarrow 1$; $j \leftarrow 1$
6. for $k \leftarrow p$ to r
7. do if $L[i] \leq R[j]$
8. then $A[k] \leftarrow L[i]$
9. $i \leftarrow i + 1$
10. else $A[k] \leftarrow R[j]$
11. $j \leftarrow j + 1$



Running Time of Merge (assume last for loop)

Initialization (copying into temporary arrays)

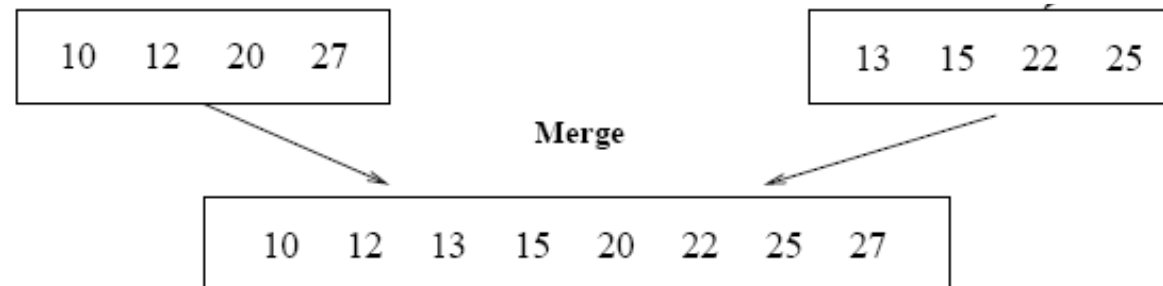
- $\Theta(n_1 + n_2) = \Theta(n)$

Adding the elements to the final array:

- n iterations, each taking constant time $\Rightarrow \Theta(n)$

Total time for Merge:

- $\Theta(n)$



Analyzing Divide-and Conquer Algorithms

The recurrence is based on the three steps of the paradigm:

- $T(n)$ – running time on a problem of size n
- **Divide** the problem into a subproblems, each of size n/b : takes $D(n)$
- **Conquer** (solve) the subproblems $aT(n/b)$
- **Combine** the solutions $C(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

}

MERGE-SORT Running Time

Divide:

- compute q as the average of p and r : $D(n) = \Theta(1)$

Conquer:

- recursively solve 2 subproblems, each of size $n/2 \Rightarrow 2T(n/2)$

Combine:

- MERGE on an n -element subarray takes $\Theta(n)$ time $\Rightarrow C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

Solve the Recurrence

Use Master's Theorem:

Compare n with $f(n) = cn$

Case 2: $T(n) = \Theta(n \lg n)$

Merge Sort - Discussion

Running time insensitive of the input

Properties:

1. Uses divide and conquer
2. It is stable
3. It is not an in-space algorithms.
4. Does not require random access of data.

Advantages:

- Guaranteed to run in $\Theta(n \lg n)$

Disadvantage

- Requires extra space $\approx N$

Sorting Files That are Almost in Order

Selection sort?

- NO, always takes quadratic time

Bubble sort?

- NO, bad for some definitions of “almost in order”
- Ex: B C D E F G H I J K L M N O P Q R S T U V W X Y Z A

Insertion sort?

- YES, takes linear time for most definitions of “almost in order”

Mergesort or custom method?

- Probably not: insertion sort simpler and faster

Recurrence for merge sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n , but only when it has no effect on the asymptotic solution to the recurrence.
- Lecture 2 provides several ways to find a good upper bound on $T(n)$.

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

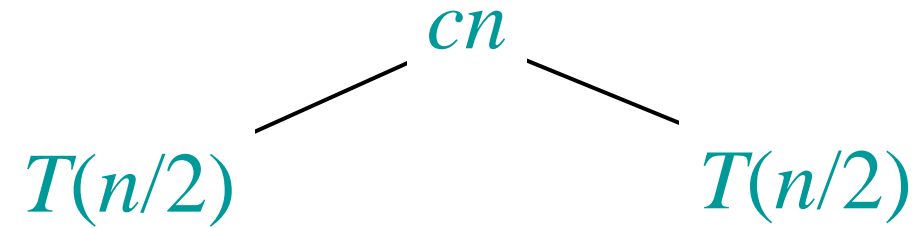
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$$T(n)$$

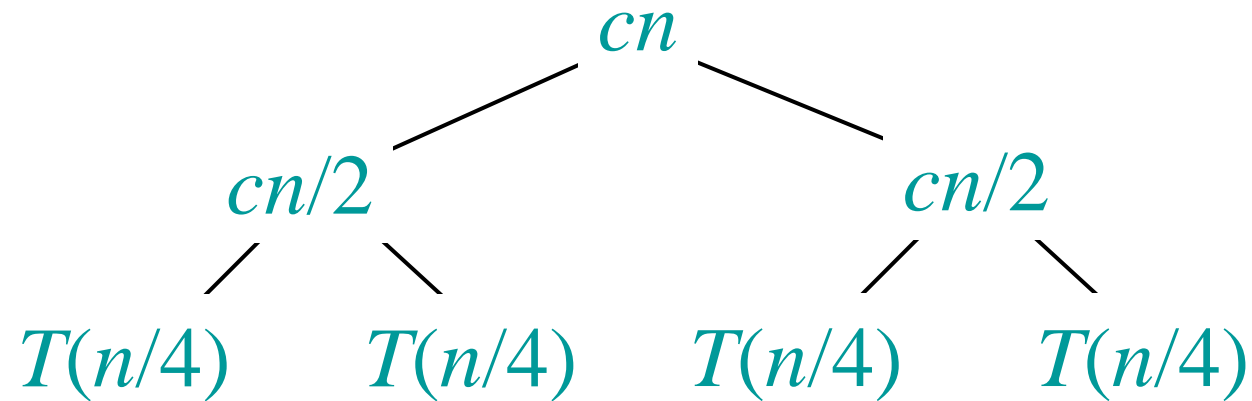
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



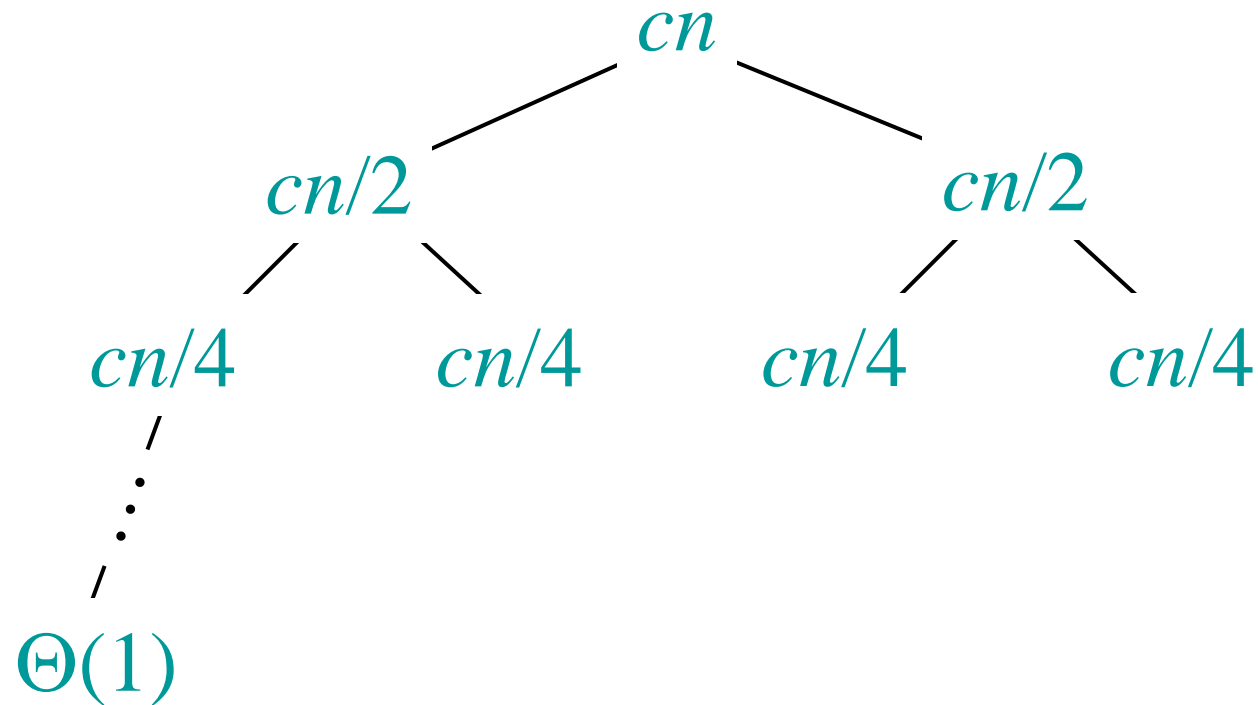
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



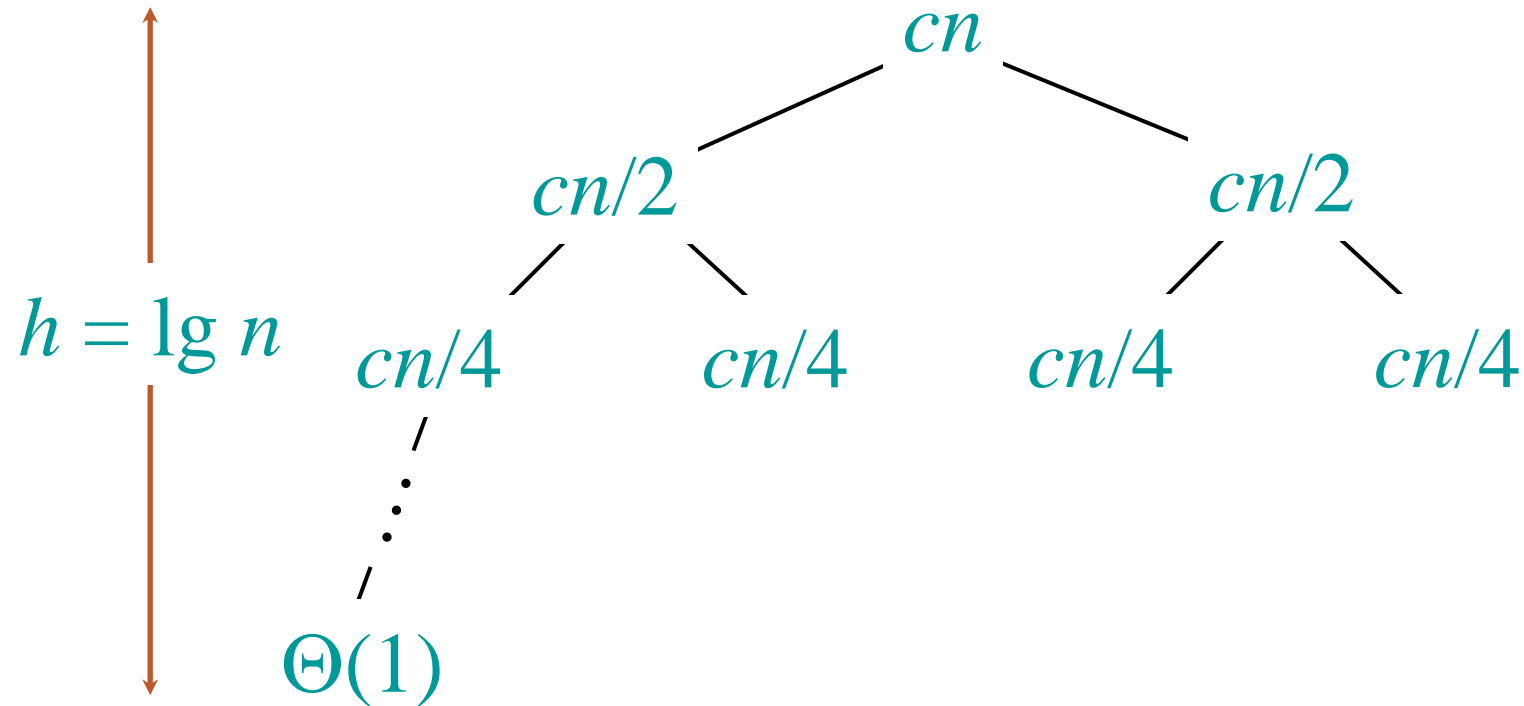
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



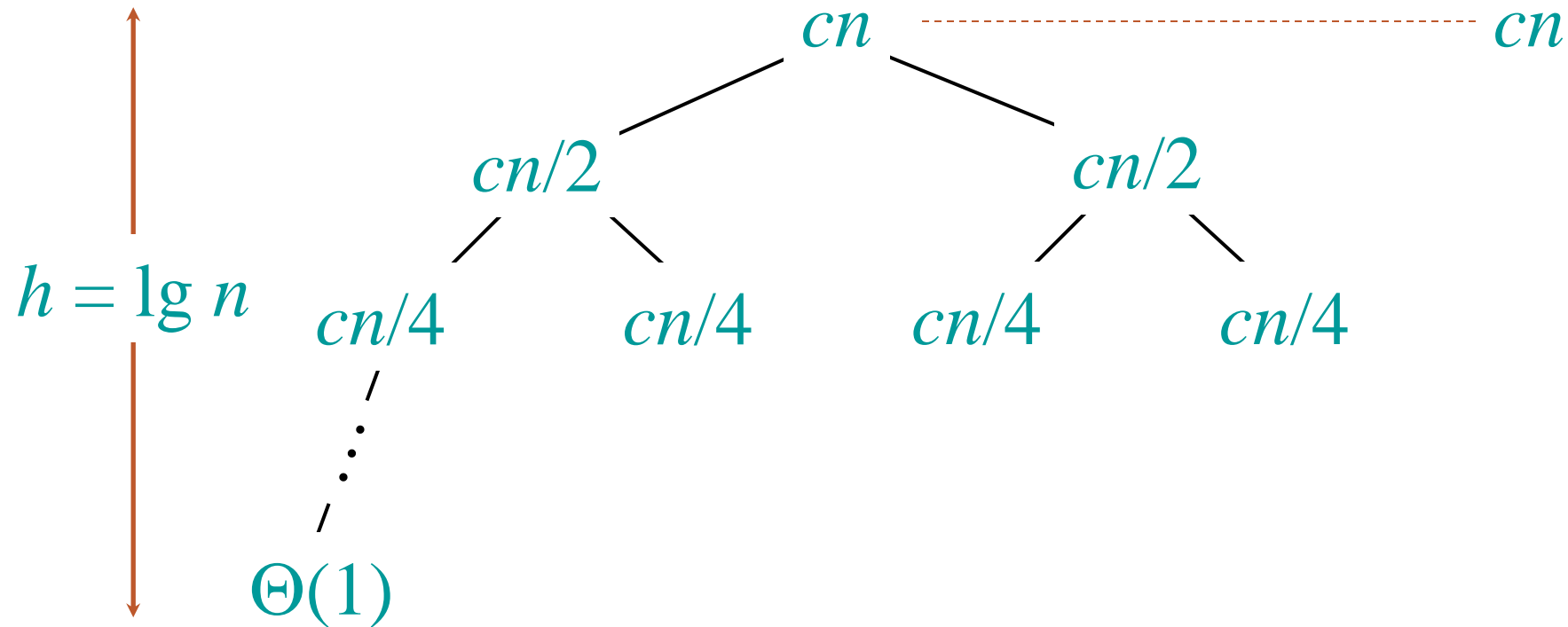
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



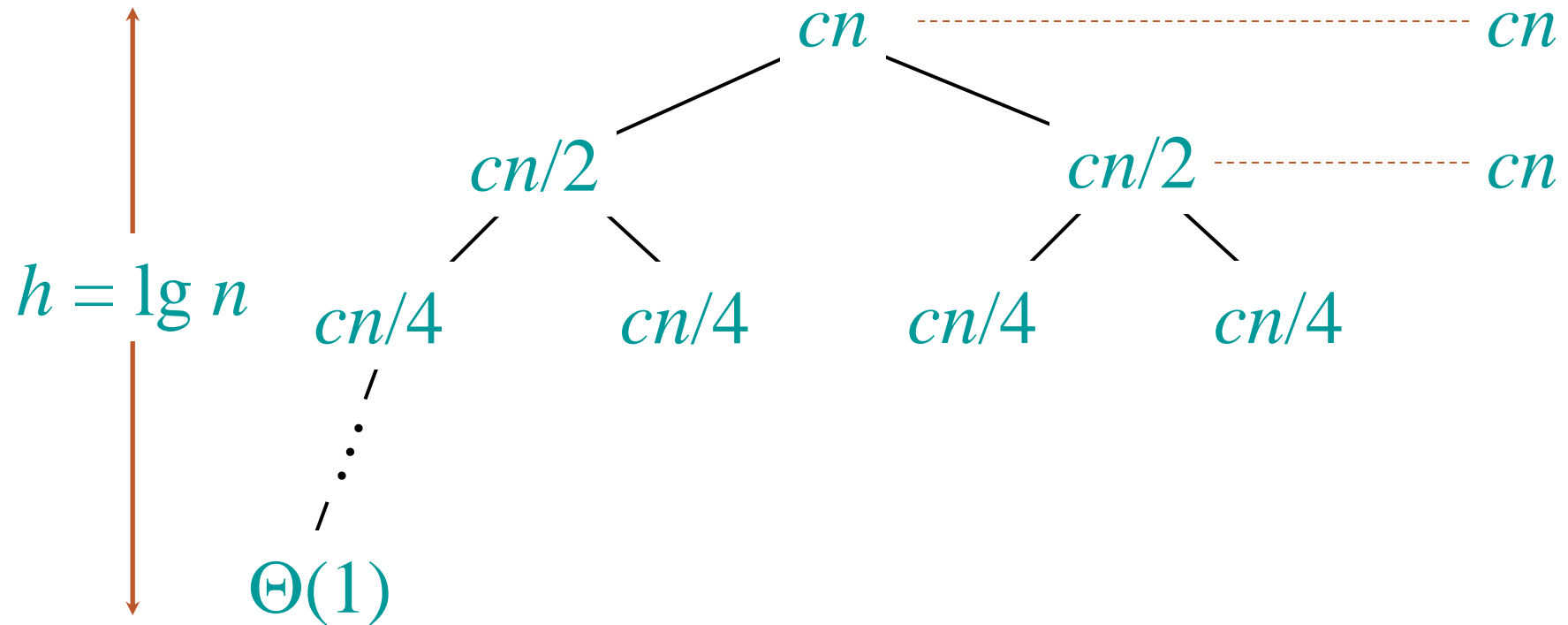
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



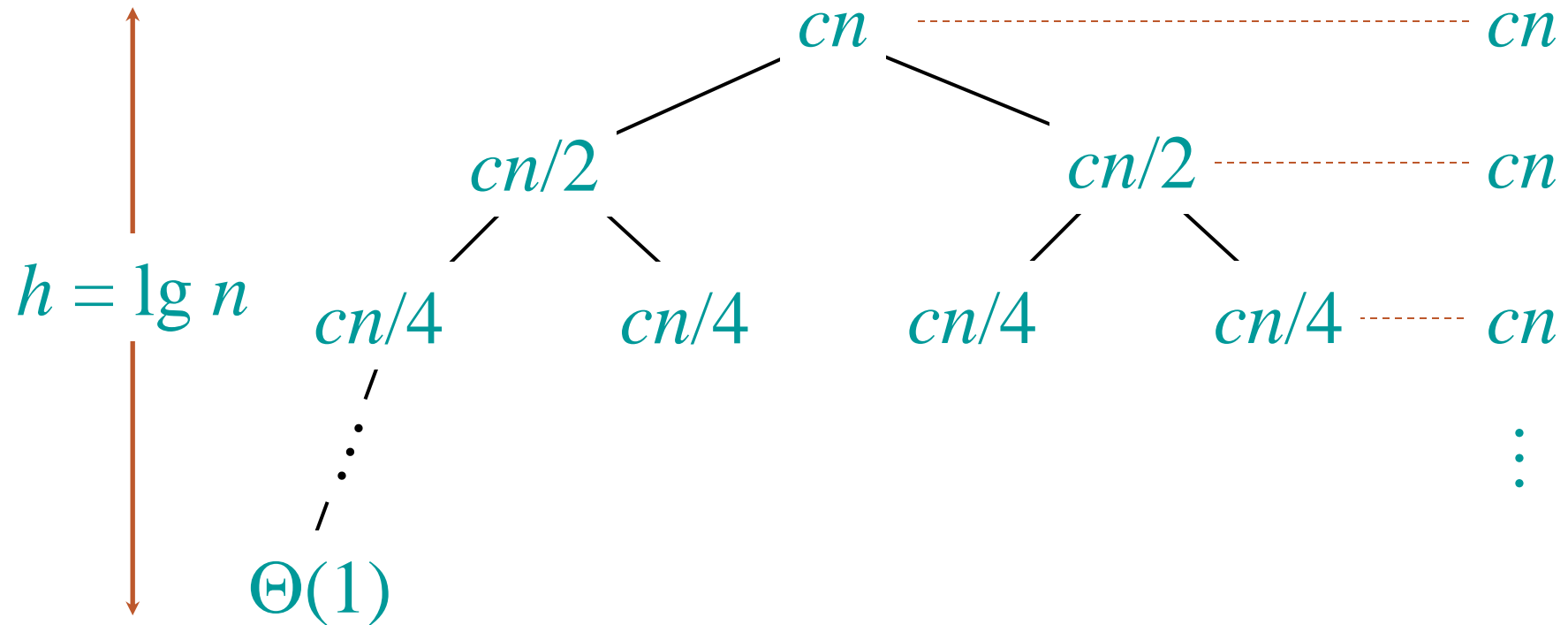
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



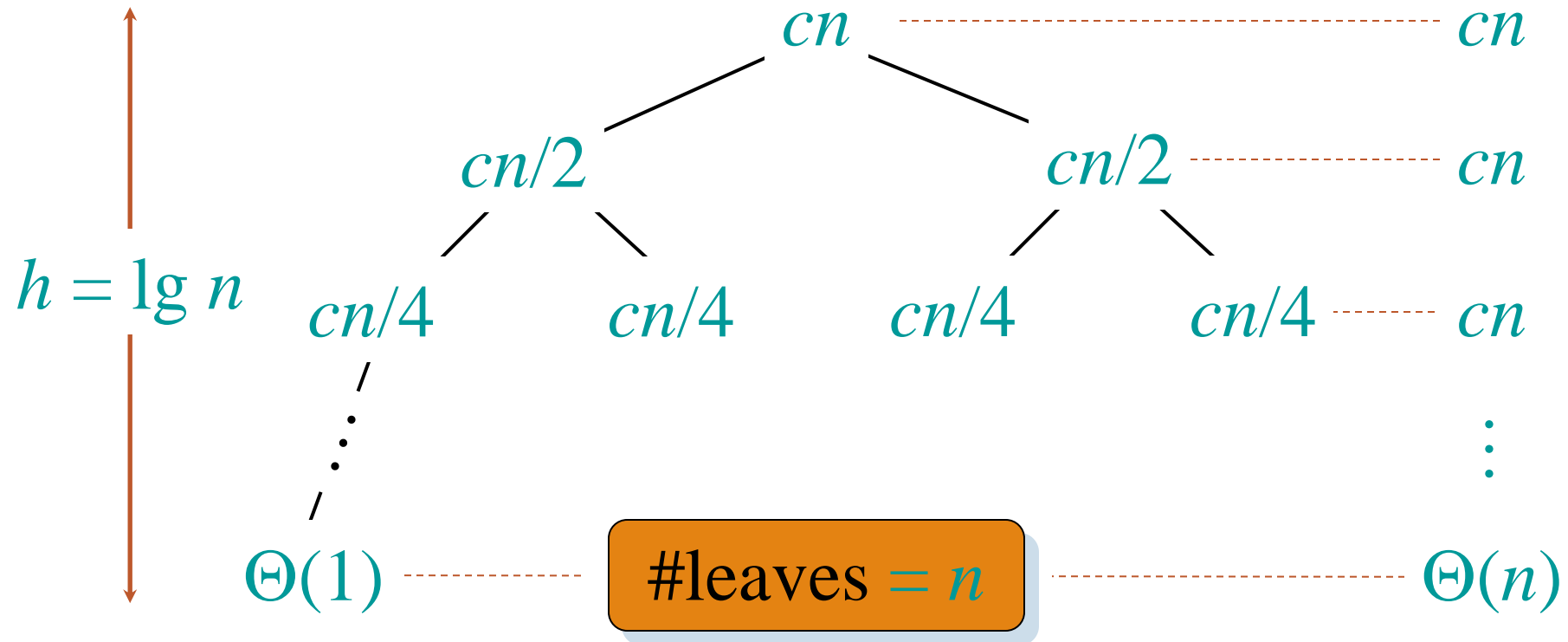
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



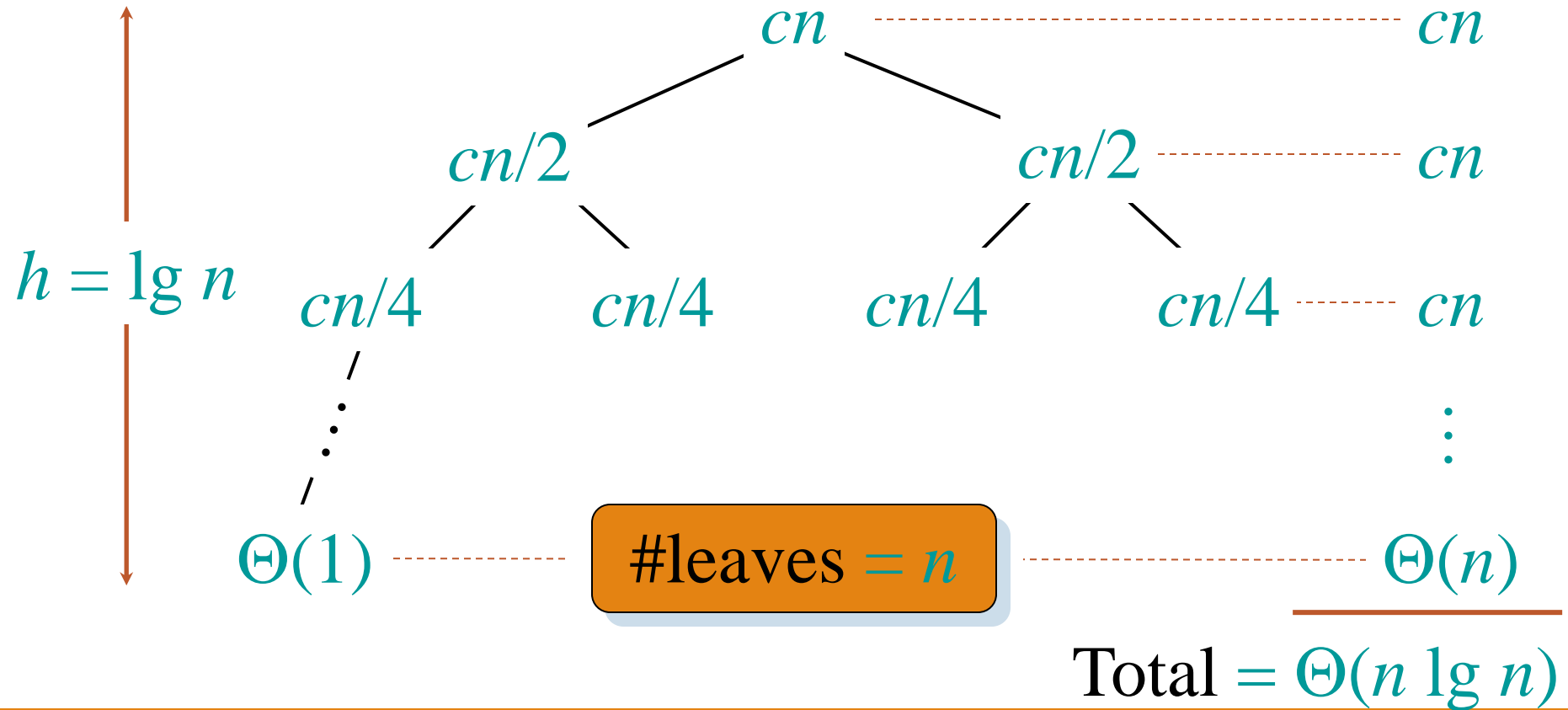
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



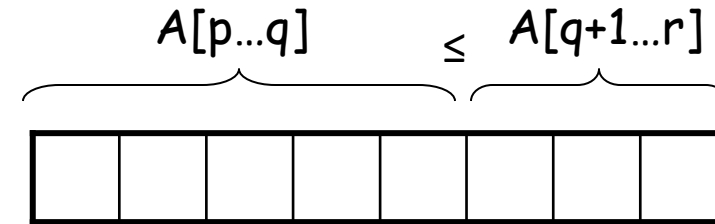
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



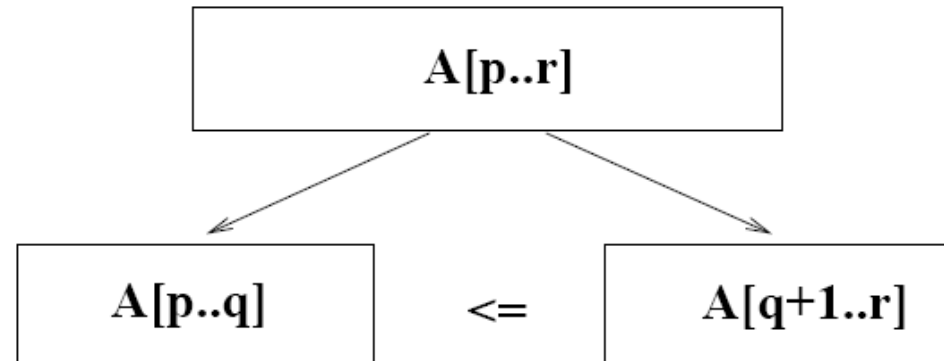
Quicksort

Sort an array $A[p..r]$

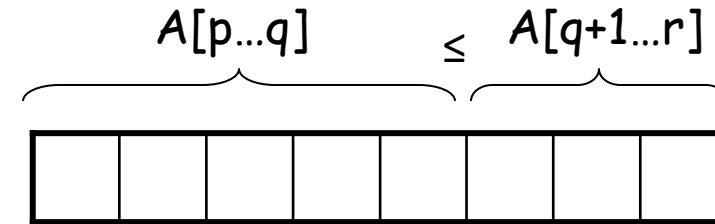


Divide

- Partition the array A into 2 subarrays $A[p..q]$ and $A[q+1..r]$, such that each element of $A[p..q]$ is smaller than or equal to each element in $A[q+1..r]$
- Need to find index q to partition the array



Quicksort



Conquer

- Recursively sort $A[p\dots q]$ and $A[q+1\dots r]$ using Quicksort

Combine

- Trivial: the arrays are sorted in place
- No additional work is required to combine them
- The entire array is now sorted

QUICKSORT

Initially: $p=1, r=n$

Alg.: QUICKSORT(A, p, r)

if $p < r$

then $q \leftarrow \text{PARTITION}(A, p, r)$

QUICKSORT (A, p, q)

QUICKSORT ($A, q+1, r$)
Recurrence:

$$T(n) = T(q) + T(n - q) + f(n)$$

$f(n)$ depends on PARTITION()

Partitioning the Array

Choosing PARTITION()

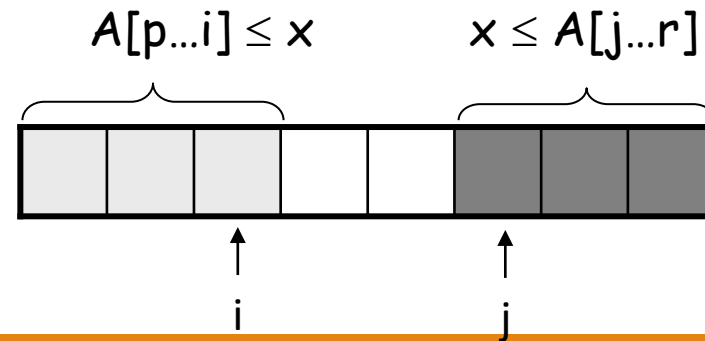
- There are different ways to do this
- Each has its own advantages/disadvantages

Hoare partition (see prob. 7-1, page 159)

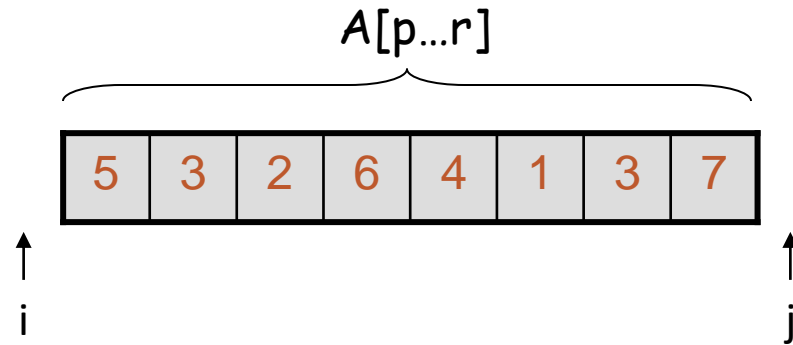
- Select a pivot element x around which to partition
- Grows two regions

$$A[p \dots i] \leq x$$

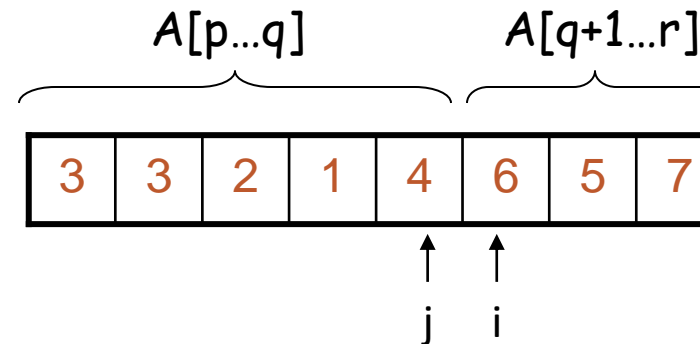
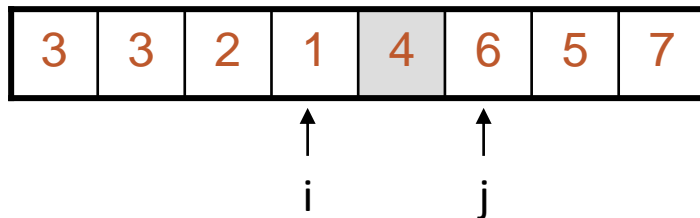
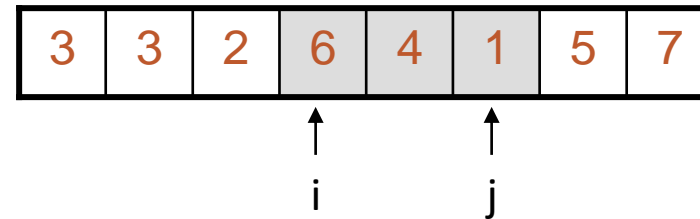
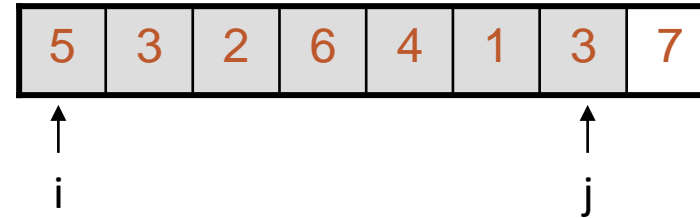
$$x \leq A[j \dots r]$$



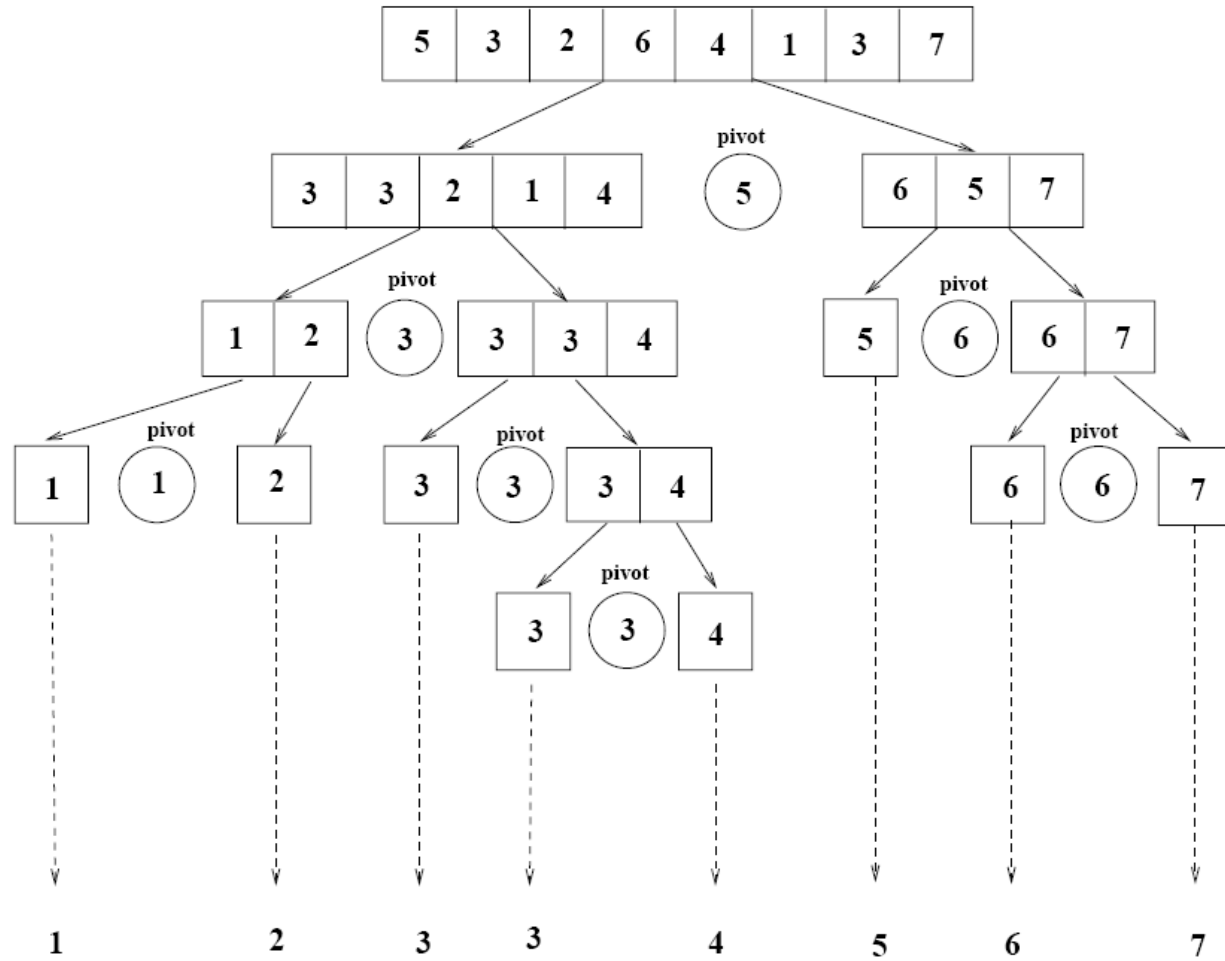
Example



pivot $x=5$



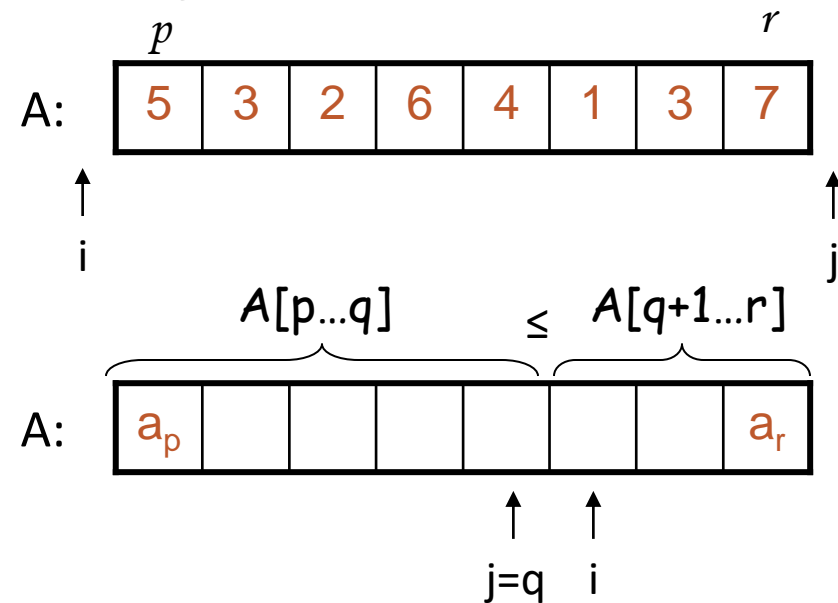
Example



Partitioning the Array

Alg. PARTITION (A, p, r)

1. $x \leftarrow A[p]$
2. $i \leftarrow p - 1$
3. $j \leftarrow r + 1$
4. **while** TRUE
5. **do repeat** $j \leftarrow j - 1$
6. **until** $A[j] \leq x$
7. **do repeat** $i \leftarrow i + 1$
8. **until** $A[i] \geq x$
9. **if** $i < j$
10. **then** exchange $A[i] \leftrightarrow A[j]$
11. **else return** j



Each element is
visited once!

Running time: $\Theta(n)$
 $n = r - p + 1$

Recurrence

Initially: $p=1, r=n$

Alg.: QUICKSORT(A, p, r)

if $p < r$

then $q \leftarrow \text{PARTITION}(A, p, r)$

QUICKSORT(A, p, q)

QUICKSORT($A, q+1, r$)

Recurrence:

$$T(n) = T(q) + T(n - q) + n$$

Worst Case Partitioning

Worst-case partitioning

- One region has one element and the other has $n - 1$ elements
- Maximally unbalanced

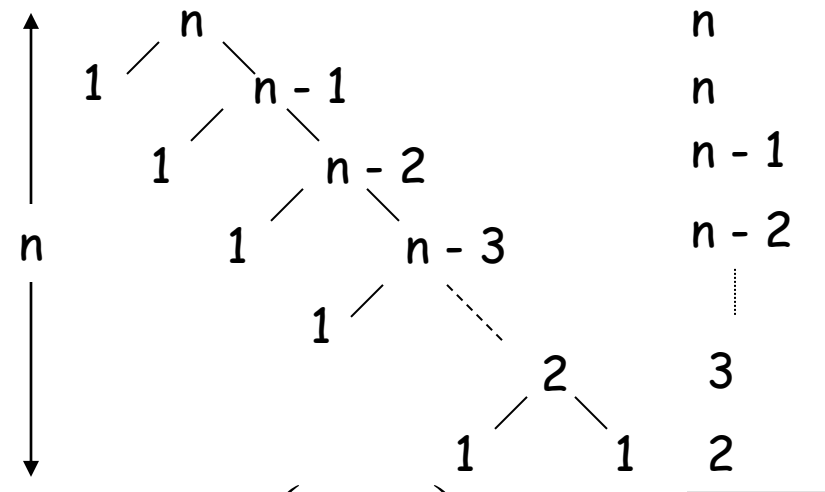
Recurrence: $q=1$

$$T(n) = T(1) + T(n - 1) + n,$$

$$T(1) = \Theta(1)$$

$$T(n) = T(n - 1) + n$$

=



$$n + \left(\sum_{k=1}^n k \right) - 1 = \Theta(n) + \Theta(n^2) = \Theta(n^2)$$

When does the worst case happen?

Best Case Partitioning

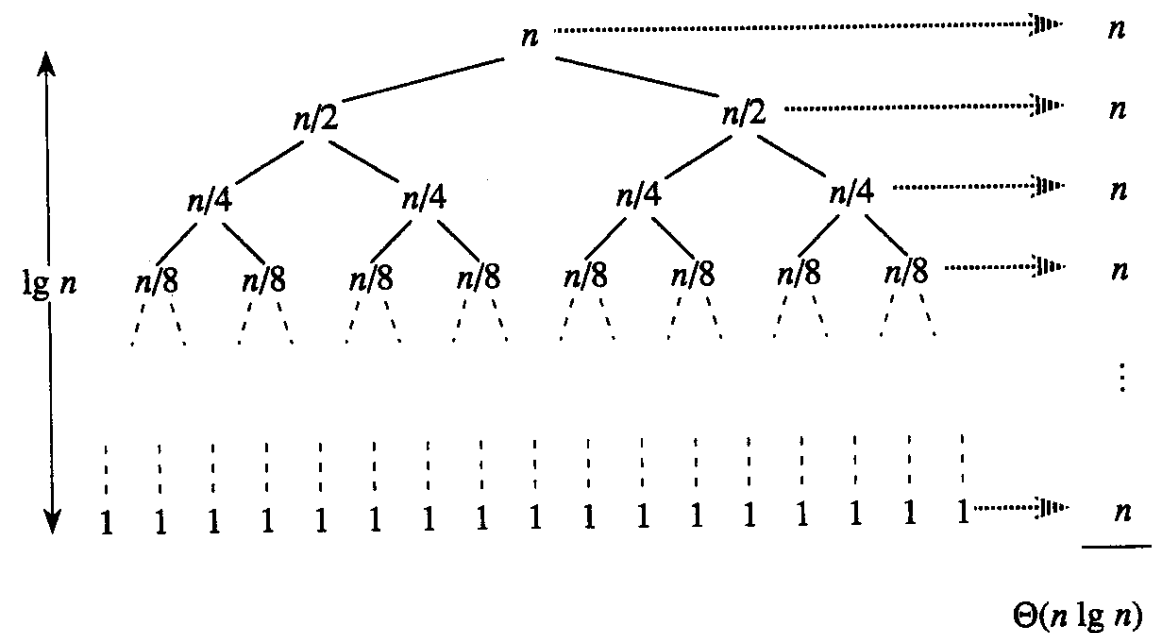
Best-case partitioning

- Partitioning produces two regions of size $n/2$

Recurrence: $q=n/2$

$$T(n) = 2T(n/2) + \Theta(n)$$

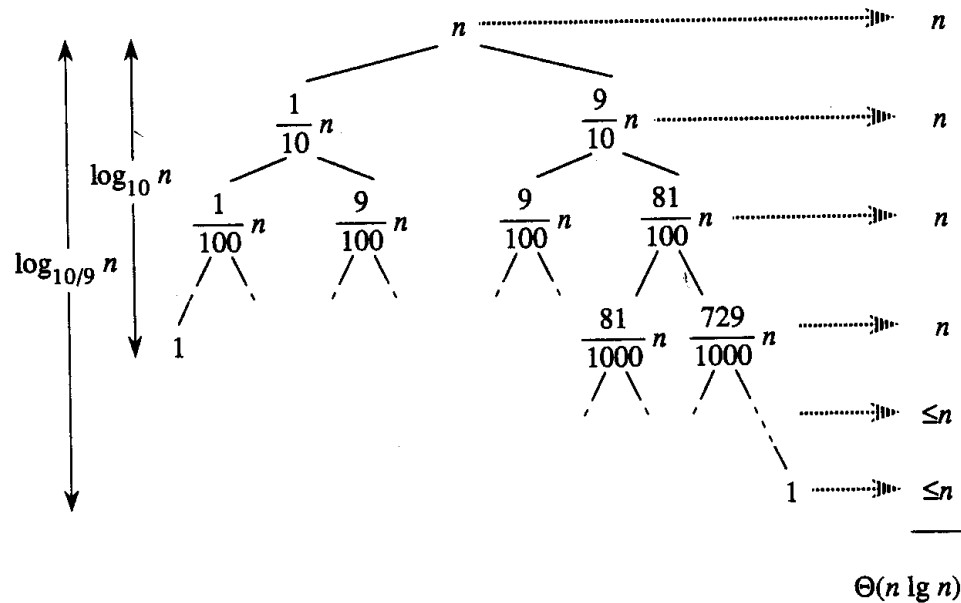
$$T(n) = \Theta(n \lg n) \text{ (Master theorem)}$$



Case Between Worst and Best

9-to-1 proportional split

$$Q(n) = Q(9n/10) + Q(n/10) + n$$



- Using the recursion tree:

$$\text{longest path: } Q(n) \leq n \sum_{i=0}^{\log_{10/9} n} 1 = n(\log_{10/9} n + 1) = c_2 n \lg n$$

$$\text{shortest path: } Q(n) \geq n \sum_{i=0}^{\log_{10} n} 1 = n \log_{10} n = c_1 n \lg n$$

$$\text{Thus, } Q(n) = \Theta(n \lg n)$$

How does partition affect performance?

- **Any splitting of constant proportionality** yields $\Theta(n \lg n)$ time !!!

- Consider the $(1 : n - 1)$ splitting:

ratio = $1/(n - 1)$ not a constant !!!

- Consider the $(n/2 : n/2)$ splitting:

ratio = $(n/2)/(n/2) = 1$ it is a constant !!

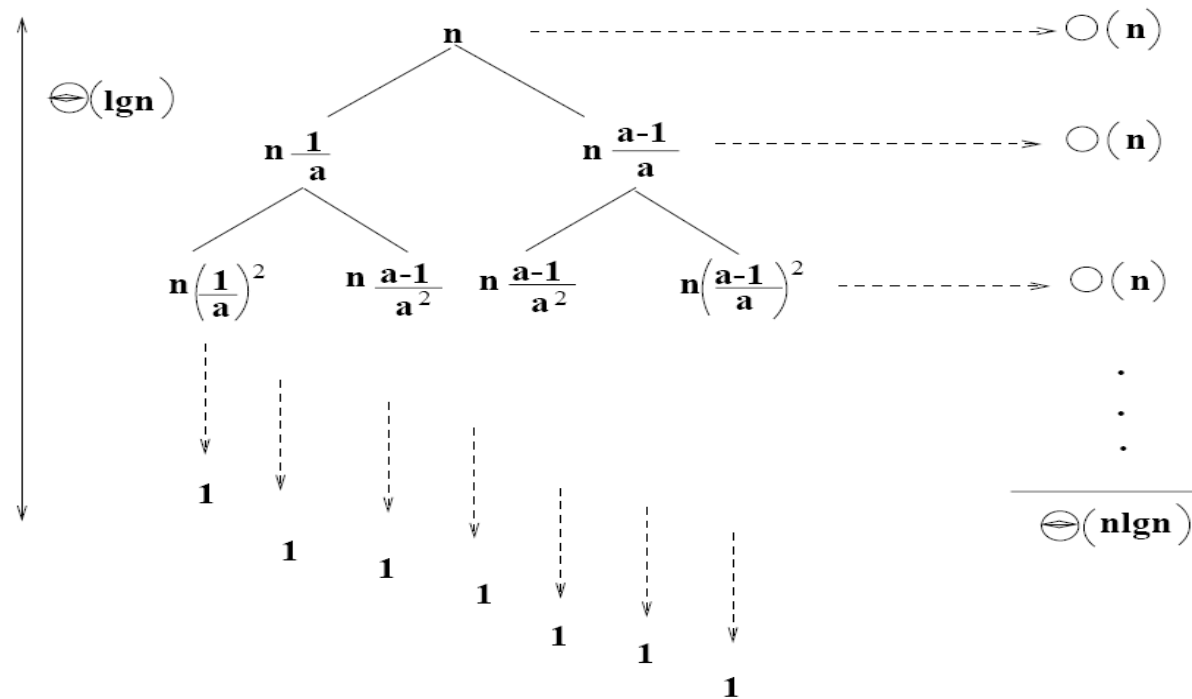
- Consider the $(9n/10 : n/10)$ splitting:

ratio = $(9n/10)/(n/10) = 9$ it is a constant !!

How does partition affect performance?

- Any $((a-1)n/a : n/a)$ splitting:

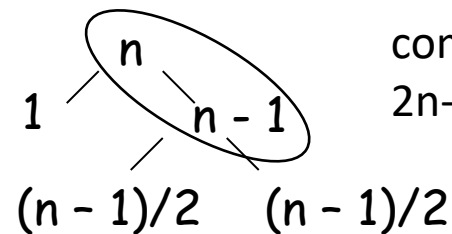
ratio $= ((a-1)n/a)/(n/a) = a-1$ it is a constant !!



Performance of Quicksort

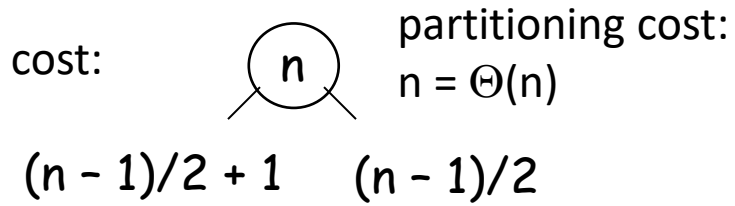
Average case

- All permutations of the input numbers are equally likely
- On a random input array, we will have a **mix** of well balanced and unbalanced splits
- Good and bad splits are randomly distributed across throughout the tree



Alternate of a good
and a bad split

combined partitioning cost:
 $2n-1 = \Theta(n)$



Nearly well
balanced split

- Running time of Quicksort when levels alternate between good and bad splits is $O(n \lg n)$

Quick Sort - Discussion

Properties:

It is an in-space algorithm

It uses divide and conquer strategy

Not a stable sorting algorithm

Time complexity: Best and average case: $O(n \log n)$

Worst case: $O(n^2)$ which can be easily avoided by using randomized partition algorithm

Master's Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k \log^p n)$$

$a \geq 1, b > 1, k \geq 0$, p is a real number then:

1) If $a > b^k$ then

$$T(n) = \Theta(n^{\log_b a})$$

2) If $a = b^k$ then

a) If $p > -1$, then

$$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$$

b) If $p = -1$, then

$$T(n) = \Theta(n^{\log_b a} \log \log n)$$

c) If $p < -1$, then

$$T(n) = \Theta(n^{\log_b a})$$

3) If $a < b^k$

a) If $p \geq 0$ then $T(n) = \Theta(n^k \log^p n)$

b) If $p < 0$, then $T(n) = \Theta(n^k)$.

$$d) T(n) = 3T\left(\frac{n}{2}\right) + n^2$$

$$a = 3, b = 2, k = 2, p = 0$$

$$a = 3, b^k = 2^2 = 4$$

$\Rightarrow a < b^k$ and $p = 0$

$$\text{we s.a. } \Rightarrow T(n) = \Theta(n^k \log^p n)$$

$$= \Theta(n^2 \log^0 n) = \Theta(n^2)$$