COMP3251
Lecture 4: Selection (Chapter 2.4)

Selection

Input: A set S of n integers $x_1, x_2, \ldots, x_n$, and an integer $1 \leq k \leq n$.

Output: The k-th smallest integer x^* among $x_1, x_2, \ldots, x_n$.

Selection

Input: A set S of n integers $x_1, x_2, \ldots, x_n$, and an integer $1 \leq k \leq n$.

Output: The k-th smallest integer x^* among $x_1, x_2, \ldots, x_n$.

A straight-forward selection algorithm:

1) Find the smallest integer. ($O(n)$ time)
2) Find the 2nd smallest integer. ($O(n-1)$ time)

   …

k) Find the k-th smallest integer. ($O(n-k+1)$ time)

Selection

Input: A set S of n integers $x_1, x_2, \ldots, x_n$, and an integer $1 \leq k \leq n$.

Output: The k-th smallest integer x^* among $x_1, x_2, \ldots, x_n$.

A straight-forward selection algorithm:

1) Find the smallest integer. ($O(n)$ time)
2) Find the 2nd smallest integer. ($O(n-1)$ time)

   …

k) Find the k-th smallest integer. ($O(n-k+1)$ time)

Running time: $O(nk)$

Selection

Input: A set S of n integers $x_1, x_2, \ldots, x_n$, and an integer $1 \le k \le n$.

Output: The k-th smallest integer x^* among $x_1, x_2, \ldots, x_n$.

Another straight-forward selection algorithm:

1) Sort the input integers in ascending order (e.g., Merge Sort); Let $y_1, \ldots, y_n$ be the sorted list.
2) Output y_k.

Example:

Select the 3rd smallest integer among $73, 44, 34, 18, 29, 27$.

1) $x_1, x_2, \ldots, x_6 = 73, 44, 34, 18, 29, 27$;
2) $y_1, y_2, \ldots, y_6 = 18, 27, 29, 34, 44, 73$;
3) $k = 3$ and $x^* = y_3 = 29$.

Selection

Input: A set S of n integers $x_1, x_2, \ldots, x_n$, and an integer $1 \le k \le n$.

Output: The k-th smallest integer x^* among $x_1, x_2, \ldots, x_n$.

Another straight-forward selection algorithm:

1) Sort the input integers in ascending order (e.g., Merge Sort); Let $y_1, \ldots, y_n$ be the sorted list.
2) Output y_k.

Example:

Select the 3rd smallest integer among *73*, *44*, *34*, *18*, *29*, *27*.

1) $x_1, x_2, \ldots, x_6 = 73, 44, 34, 18, 29, 27$;
2) $y_1, y_2, \ldots, y_6 = 18, 27, 29, 34, 44, 73$;
3) $k = 3$ and $x^* = y_3 = 29$.

Running time: $O(n \log n)$

A Divide and Conquer Selection Algorithm

Select(S, k)

Divide: Pick an arbitrary value v among $x_1, x_2, \ldots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse: Recurse on the subset that contains x^*.

rank	1	2	3	…		…		…		…	
integer					v	v	v				

S_L S_v S_R

Note: The ranks are for explanation only; we don't need to sort the numbers.

A Divide and Conquer Selection Algorithm

Select(S, k)

Divide: Pick an arbitrary value v among $x_1, x_2, \dots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse: Recurse on the subset that contains x^*.

rank	1	2	3	…		…		…		…	
integer					v	v	v				

S_L S_v S_R

Note: The ranks are for explanation only; we don't need to sort the numbers.

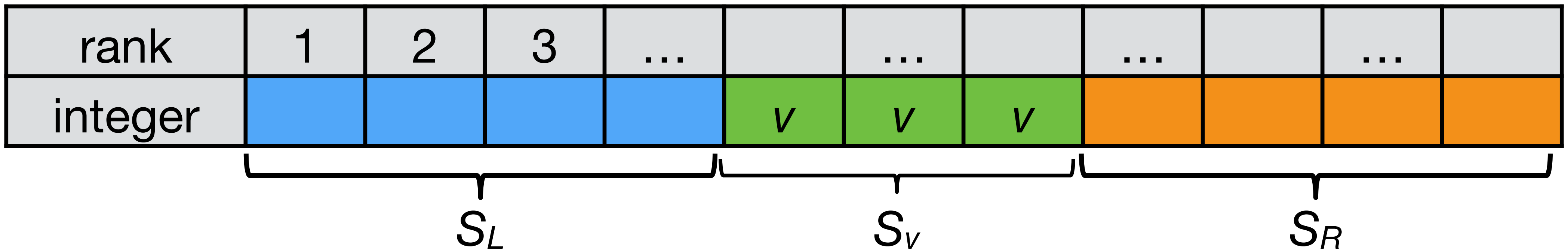
Key questions: Which subset contains x^*? What is the rank of x^* in the subset?

A Divide and Conquer Selection Algorithm

rank	1	2	3	...		...		...		...	
integer					v	v	v				

S_L S_v S_R

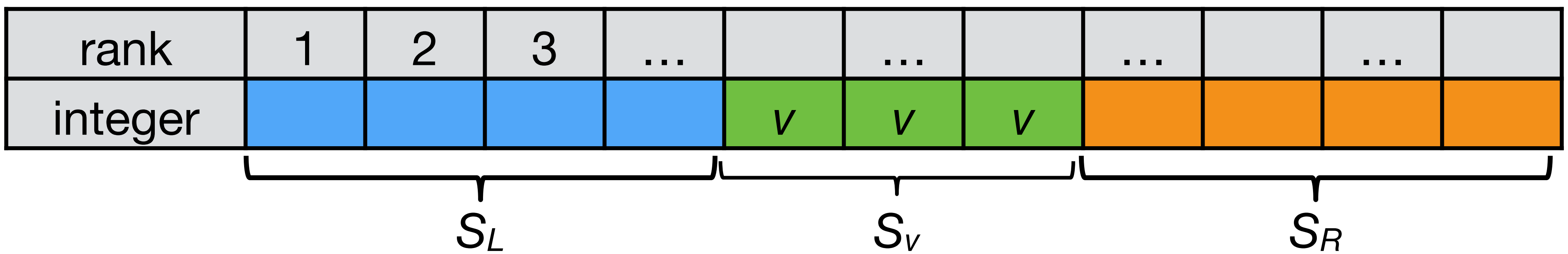
A Divide and Conquer Selection Algorithm



Case 1: $k \le |S_L|$.

- x^* is the k-th smallest integer in S_L.
- Output **Select**(S_L, k).

A Divide and Conquer Selection Algorithm



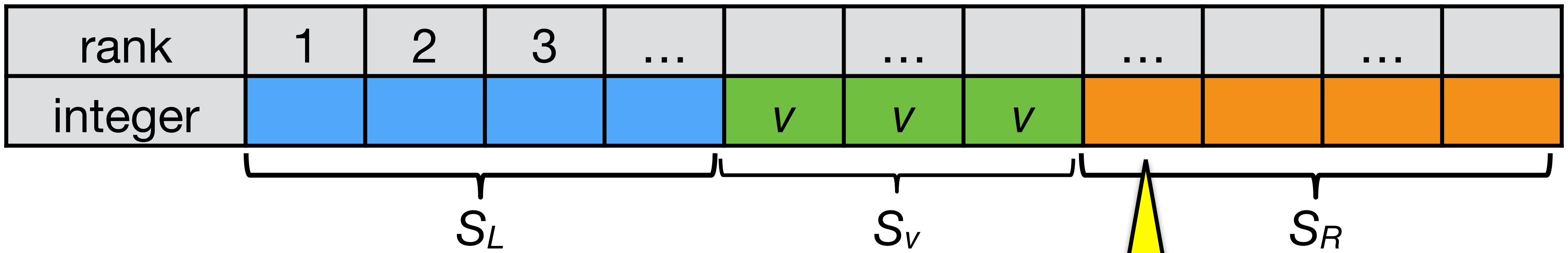
Case 1: $k \leq |S_L|$.

- x^* is the k-th smallest integer in S_L.
- Output **Select**(S_L, k).

Case 2: $|S_L| < k \leq |S_L| + |S_v|$.

- x^* is in S_v.
- Output v. (Note that all numbers in S_v are equal to v.)

A Divide and Conquer Selection Algorithm



The *1*st number in S_R is the $(|S_L| + |S_v| + 1)$-th number in S.

So the i-th number in S_R is the $(|S_L| + |S_v| + i)$-th number in S.

Thus x* is the $(k - |S_L| - |S_v|)$-th smallest integer in S_R.

Case 1: $k \le |S_L|$.
- x^* is the k-th smallest integer in S_L.
- Output **Select**(S_L, k).

Case 2: $|S_L| < k \le |S_L| + |S_v|$.
- x^* is in S_v.
- Output v. (Note that all numbers in S_v are equal to v.)

Case 3: $|S_L| + |S_v| < k \le n$.
- x^* is the $(k - |S_L| - |S_v|)$-th smallest integer in S_R.
- Output **Select**(S_R, $k - |S_L| - |S_v|$).

A Divide and Conquer Selection Algorithm

Select(S, k)

Divide: Pick an arbitrary value v among $x_1, x_2, \dots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse:
1) If $k \le |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \le |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S	73	44	34	18	29	27

level *1*

S_L S_v S_R

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, k - $|S_L|$ - $|S_v|$).

Sample Run

S

73	44	34	18	29	**27**

level *1*

S_L S_v S_R

1) If $k \le |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \le |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S

73	44	34	18	29	27

level *1*

27

S_L S_v S_R

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S

73	44	34	18	29	27

level *1*

27

S_L S_v S_R

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S

73	44	34	18	29	27

level *1*

27 **73**

S_L S_v S_R

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S	73	**44**	34	18	29	27

level *1*

S_L	S_v	S_R
	27	**73**

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

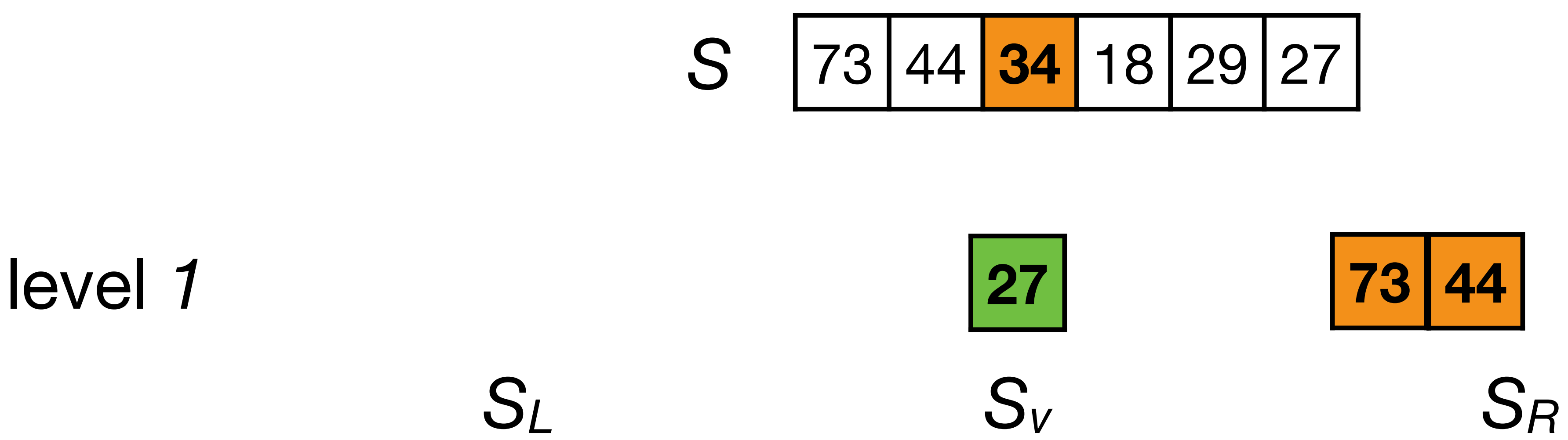
S	73	44	34	18	29	27

level *1*

	S_L	S_v	S_R
		27	73, 44

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

Sample Run



1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

Sample Run

S						
	73	44	34	18	29	27

level *1*

S_L	S_v	S_R
	27	**73** **44** **34**

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

Sample Run

S | 73 | 44 | 34 | **18** | 29 | 27 |

level *1*

S_L	S_v	S_R
	27	**73** **44** **34**

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S | 73 | 44 | 34 | 18 | 29 | 27 |

level *1*

S_L	S_v	S_R
18	**27**	**73 44 34**

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, k - $|S_L|$ - $|S_v|$).

Sample Run

S | 73 | 44 | 34 | 18 | **29** | 27

level *1*

S_L	S_v	S_R
18	**27**	**73** **44** **34**

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S | 73 | 44 | 34 | 18 | 29 | 27 |

level *1*

S_L	S_v	S_R
18	**27**	**73 44 34 29**

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S: 73 | 44 | 34 | 18 | 29 | 27

level *1*

S_L: 18

S_v: 27

S_R: 73 | 44 | 34 | 29

- $|S_L| = 1$
- $|S_v| = 1$
- $k = 3 > |S_L| + |S_v|$ and $k - |S_L| + |S_v| = 1$
- So x* is the 1st number is S_R.

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S

73	44	34	18	29	27

level *1*

18

27

S'

73	44	34	29

level *2*

S_L S_v S_R

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S | 73 | 44 | 34 | 18 | 29 | 27 |

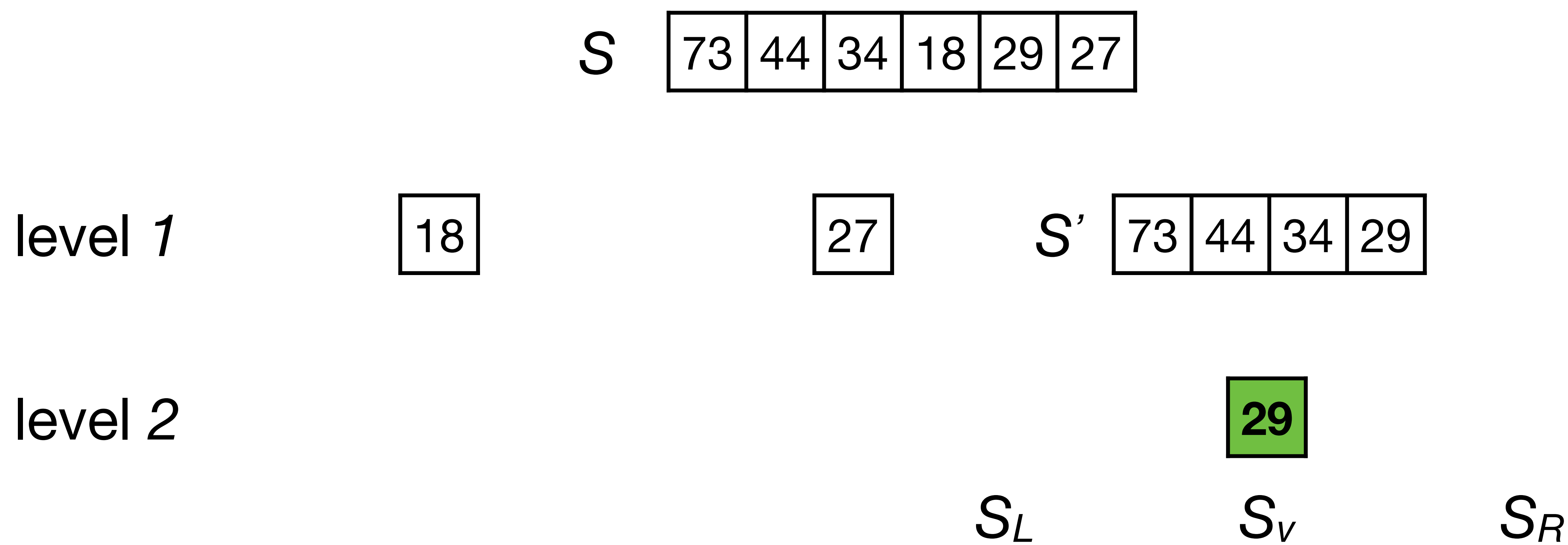
level *1* | 18 | | 27 | S' | 73 | 44 | 34 | **29** |

level *2*

S_L S_v S_R

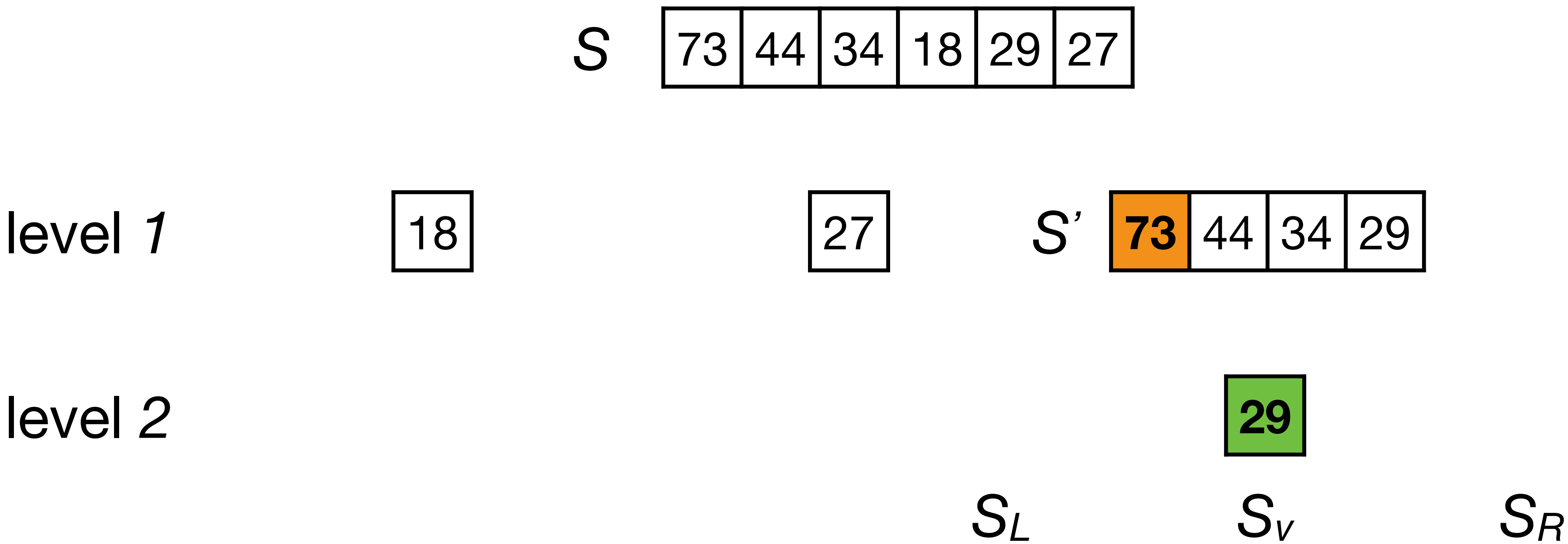
1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run



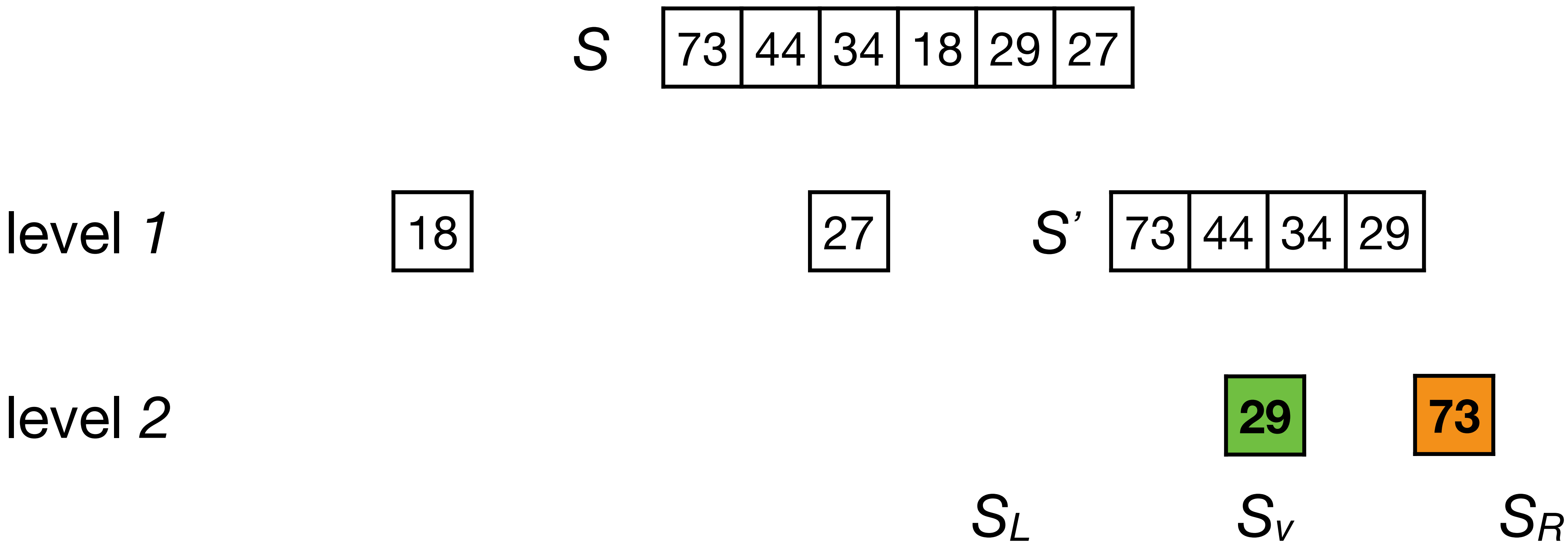
1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

Sample Run



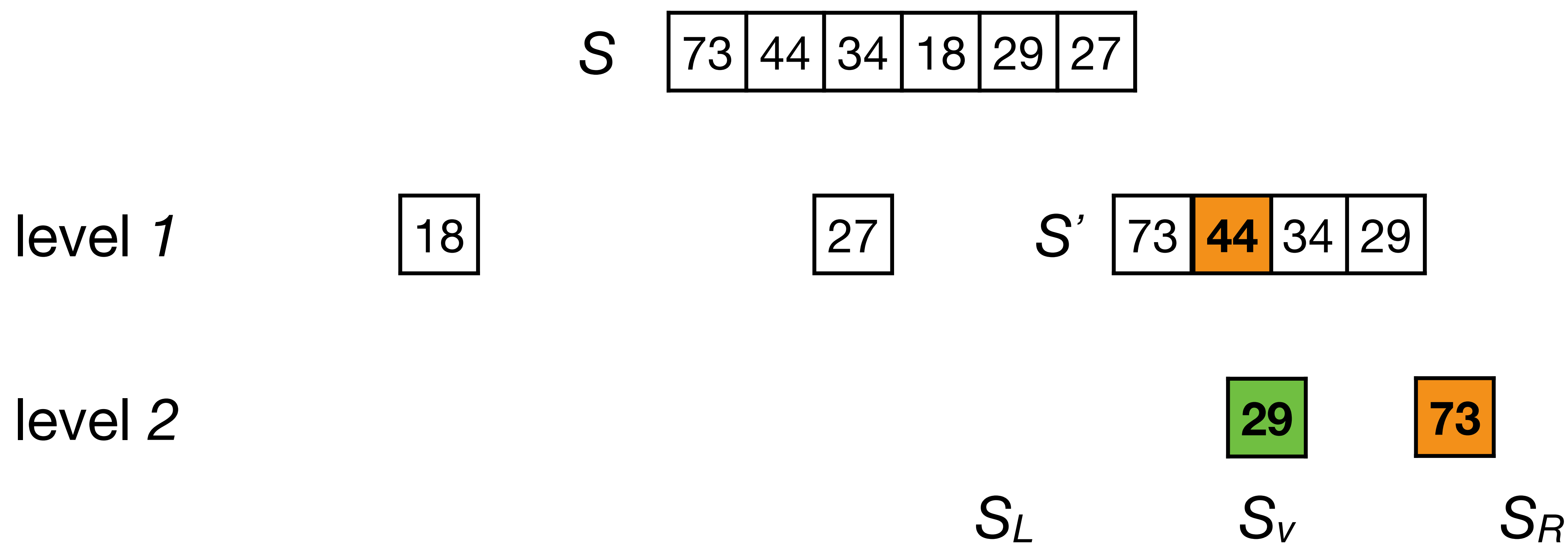
1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run



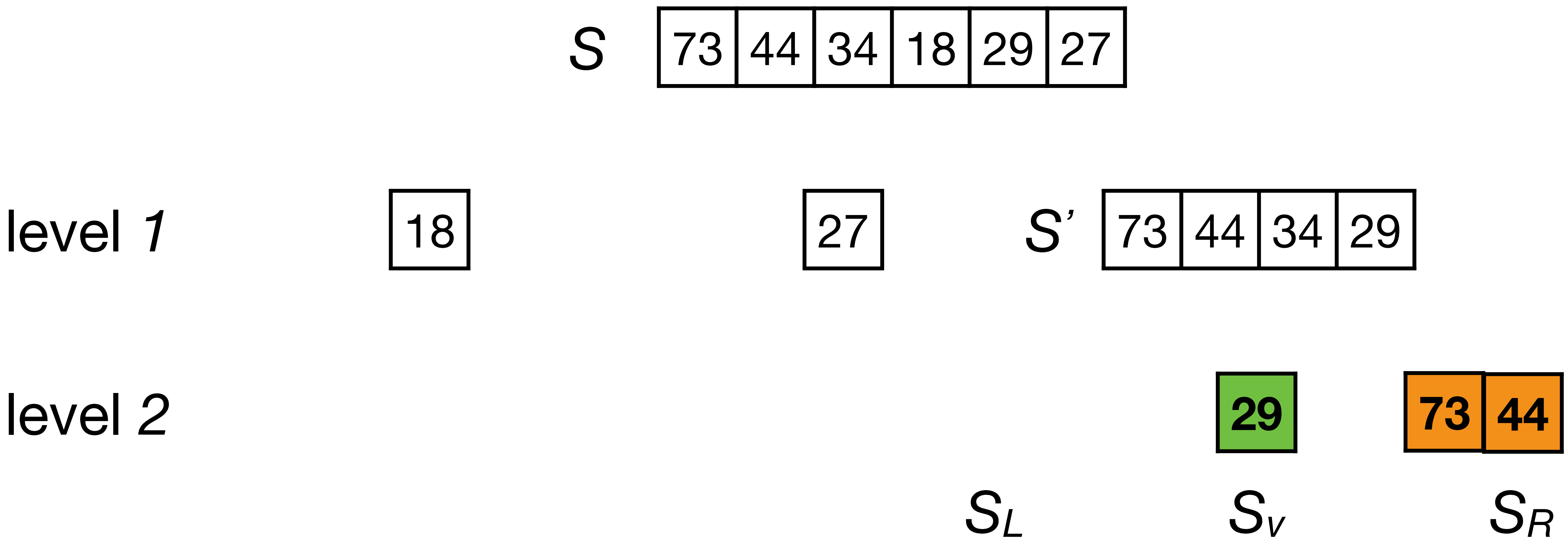
1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run



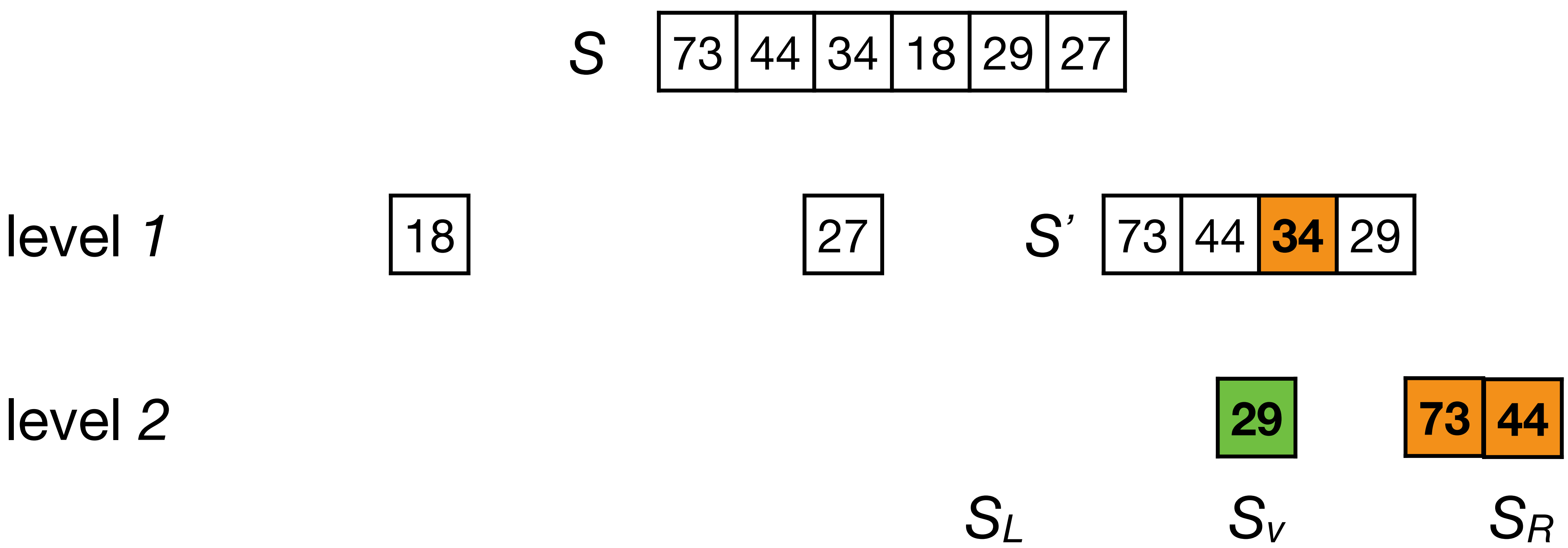
1) If $k \le |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \le |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run



1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

Sample Run



1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

Sample Run

S | 73 | 44 | 34 | 18 | 29 | 27 |

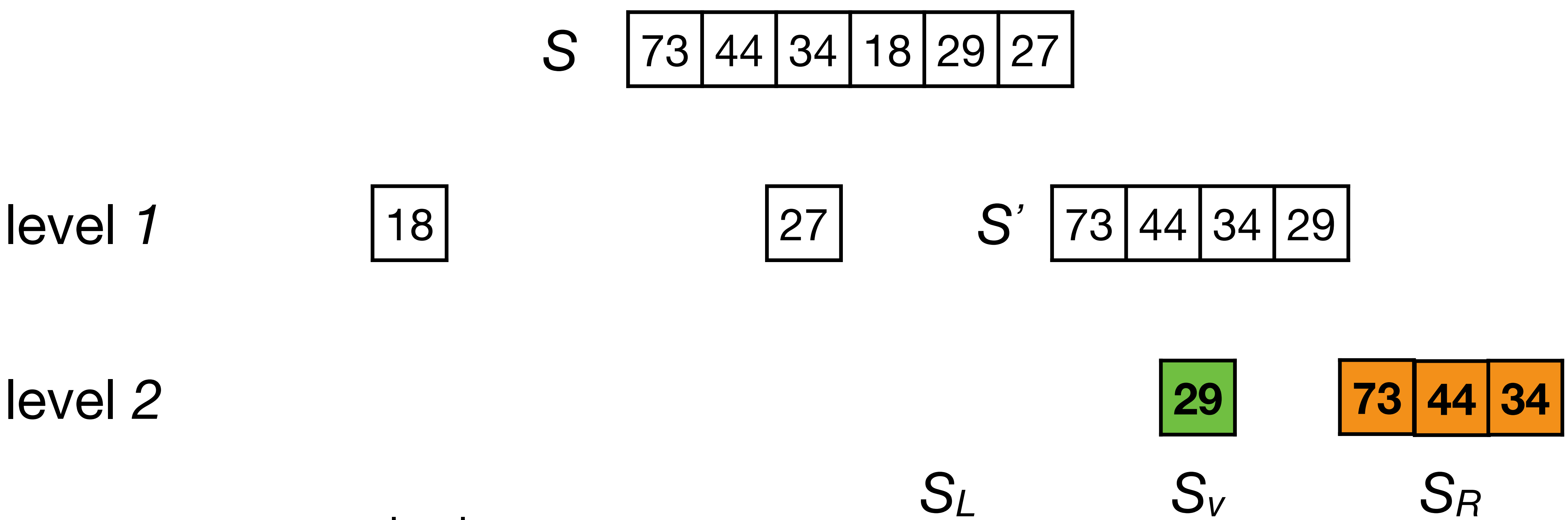
level *1* | 18 | | 27 | S' | 73 | 44 | 34 | 29 |

level *2* | **29** | | **73** | **44** | **34** |

S_L S_v S_R

1) If $k \le |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \le |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

Sample Run



- $|S_L| = 0$
- $|S_v| = 1$
- $|S_L| < k = 1 \leq |S_L| + |S_v|$
- So x* is in S_v and output $x^* = v = 29$.

1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

What is the running time?

Divide: Pick an arbitrary value v among $x_1, x_2, \dots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse:
1) If $k \le |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \le |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

What is the running time?

Divide: Pick an arbitrary value v among $x_1, x_2, \ldots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse:
1) If $k \le |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \le |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

- **Divide** takes $O(n)$ time.

What is the running time?

Divide: Pick an arbitrary value v among $x_1, x_2, \ldots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse:
1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**$(S_R, k - |S_L| - |S_v|)$.

- **Divide** takes $O(n)$ time.
- **Recurse** takes either $T(|S_L|)$ time (case 1), or $O(1)$ time (case 2), or $T(|S_R|)$ time (case 3).

What is the running time?

Divide: Pick an arbitrary value v among $x_1, x_2, \ldots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse:
1) If $k \leq |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \leq |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

- **Divide** takes $O(n)$ time.
- **Recurse** takes either $T(|S_L|)$ time (case 1), or $O(1)$ time (case 2), or $T(|S_R|)$ time (case 3).
- Note that both $|S_L|$ and $|S_R|$ are at most n-1. So we have:

$$T(n) \leq T(n-1) + O(n) \leq T(n-2) + O(n-1) + O(n) = \ldots = O(1 + 2 + \ldots + n) = O(n^2)$$

Is it really that bad?

Is it really that bad?

- Yes, if we are extremely unlucky, and pick either the smallest or largest number as *v* in every round.

Is it really that bad?

- Yes, if we are extremely unlucky, and pick either the smallest or largest number as *v* in every round.
- But, if we are extremely lucky, and pick the *n/2* smallest every time, then

Is it really that bad?

- Yes, if we are extremely unlucky, and pick either the smallest or largest number as v in every round.
- But, if we are extremely lucky, and pick the $n/2$ smallest every time, then

$$T(n) = T(n/2) + O(n) = T(n/4) + O(n/2) + O(n) = \ldots = O(n).$$

Is it really that bad?

- Yes, if we are extremely unlucky, and pick either the smallest or largest number as *v* in every round.
- But, if we are extremely lucky, and pick the *n/2* smallest every time, then

$$T(n) = T(n/2) + O(n) = T(n/4) + O(n/2) + O(n) = \dots = O(n).$$

- In reality, we cannot be that lucky; on the other hand, we may not be that unlucky to always pick the extreme numbers.

Is it really that bad?

- Yes, if we are extremely unlucky, and pick either the smallest or largest number as *v* in every round.
- But, if we are extremely lucky, and pick the *n/2* smallest every time, then

$$T(n) = T(n/2) + O(n) = T(n/4) + O(n/2) + O(n) = \dots = O(n).$$

- In reality, we cannot be that lucky; on the other hand, we may not be that unlucky to always pick the extreme numbers.

Idea: Choose v randomly and hope that v is "reasonably good" on average?

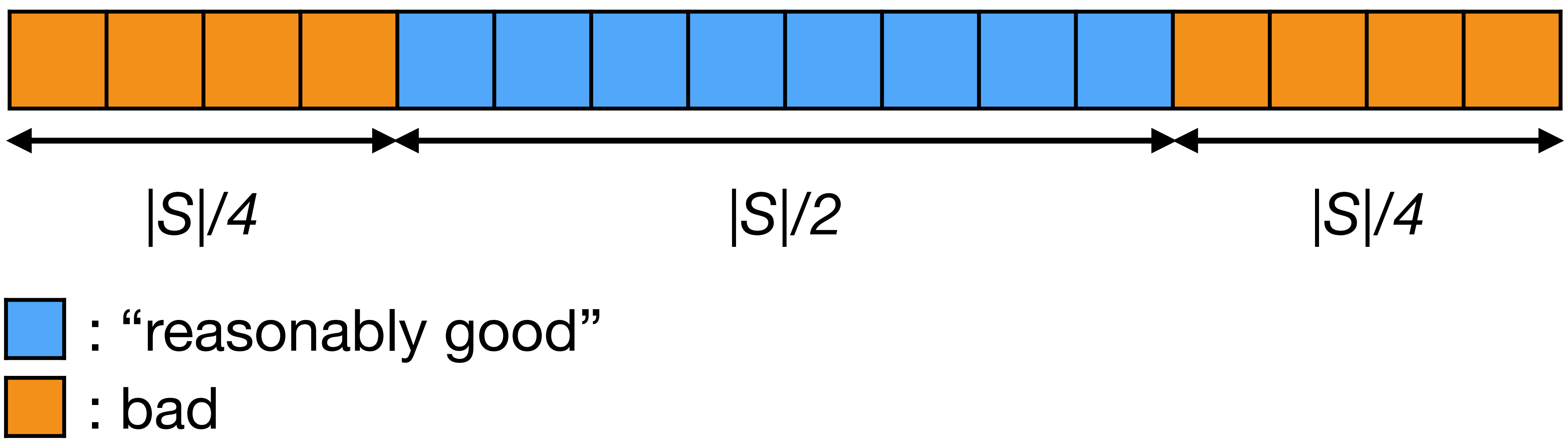
A Randomized Divide and Conquer Selection Algorithm

Select(S, k)

Divide: Pick value v randomly among $x_1, x_2, \dots, x_n$;
Divide the input integers into three subsets:
- S_L: The subset of integers that are smaller than v;
- S_v: The subset of integers that are equal to v;
- S_R: The subset of integers that are greater than v;

Recurse:
1) If $k \le |S_L|$, output **Select**(S_L, k).
2) If $|S_L| < k \le |S_L| + |S_v|$, output v.
3) If $|S_L| + |S_v| < k$, output **Select**(S_R, $k - |S_L| - |S_v|$).

What is “reasonably good”?



Observation 1: In each round, v is “reasonably good” with probability $1/2$.

Observation 2: If v is always “reasonably good”, then both $|S_L|$ and $|S_R|$ are at most $3n/4$, and $T(n) = T(3n/4) + O(n) = O(n)$!!

Expected Running Time Analysis

In each round, v is "reasonably good" with probability $1/2$.

Expected Running Time Analysis

In each round, *v* is "reasonably good" with probability *1/2*.

$$\mathbf{E}\,T(n) \;\le\; \frac{1}{2}\left(\underbrace{\mathbf{E}\,T\left(\frac{3n}{4}\right)}_{\text{"reasonably good" pivot}} + \underbrace{\mathbf{E}\,T\left(n\right)}_{\text{"bad" pivot}}\right) + O(n)$$

Expected Running Time Analysis

In each round, *v* is “reasonably good” with probability *1/2*.

$$\mathbf{E}\,T(n) \;\le\; \frac{1}{2}\left(\underbrace{\mathbf{E}\,T\!\left(\frac{3n}{4}\right)}_{\text{“reasonably good” pivot}} + \underbrace{\mathbf{E}\,T\!\left(n\right)}_{\text{“bad” pivot}}\right) + O(n)$$

- Rearranging all $\mathbf{E}\,T(n)$ terms to the left:

$$\mathbf{E}\,T(n) \;\le\; \mathbf{E}\,T\!\left(\frac{3n}{4}\right) + O(n)$$

Expected Running Time Analysis

In each round, *v* is "reasonably good" with probability *1/2*.

$$\mathbf{E}\,T(n) \;\le\; \frac{1}{2}\left(\underbrace{\mathbf{E}\,T\left(\frac{3n}{4}\right)}_{\text{"reasonably good" pivot}} + \underbrace{\mathbf{E}\,T\left(n\right)}_{\text{"bad" pivot}}\right) + O(n)$$

- Rearranging all $\mathbf{E}\,T(n)$ terms to the left:

$$\mathbf{E}\,T(n) \;\le\; \mathbf{E}\,T\left(\frac{3n}{4}\right) + O(n)$$

- Viewing $\mathbf{E}\,T(n)$ as a function of *n*, Master Theorem gives:

$$\mathbf{E}\,T(n) \;=\; O(n)$$

Alternative Analysis (from Textbook)

Alternative Analysis (from Textbook)

- In general, we can reason about the running time as follows:

$$T(n) = (\text{time to reduce the array to} \leq 3n/4) + T(3n/4)$$

Alternative Analysis (from Textbook)

- In general, we can reason about the running time as follows:

  $T(n)$ = (time to reduce the array to ≤ $3n/4$) + $T(3n/4)$

- The expected (average) running time is

$$E[\,T(n)\,] = E[\,(\text{time to reduce the array to} \le 3n/4) + T(3n/4)\,]$$

$$= E[\,(\text{time to reduce the array to} \le 3n/4)\,] + E[\,T(3n/4)\,]$$

Alternative Analysis (from Textbook)

- In general, we can reason about the running time as follows:

$$T(n) = (\text{time to reduce the array to} \le 3n/4) + T(3n/4)$$

- The expected (average) running time is

$$E[\,T(n)\,] = E[\,(\text{time to reduce the array to} \le 3n/4) + T(3n/4)\,]$$

$$= E[\,(\text{time to reduce the array to} \le 3n/4)\,] + E[\,T(3n/4)\,]$$

- Since *v* is "reasonably good" with probability *1/2* each round, by Discrete Mathematics, the algorithm gets a "reasonably good" *v* in every two rounds on average.

Alternative Analysis (from Textbook)

- In general, we can reason about the running time as follows:

$$T(n) = (\text{time to reduce the array to} \le 3n/4) + T(3n/4)$$

- The expected (average) running time is

$$E[\,T(n)\,] = E[\,(\text{time to reduce the array to} \le 3n/4) + T(3n/4)\,]$$
$$= E[\,(\text{time to reduce the array to} \le 3n/4)\,] + E[\,T(3n/4)\,]$$

- Since v is "reasonably good" with probability $1/2$ each round, by Discrete Mathematics, the algorithm gets a "reasonably good" v in every two rounds on average.
- Since process one v takes $O(n)$ time, we conclude that

$$E[\,(\text{time to reduce the array to} \le 3n/4)\,] = O(n)$$

Alternative Analysis (from Textbook)

- In general, we can reason about the running time as follows:

$$T(n) = (\text{time to reduce the array to} \leq 3n/4) + T(3n/4)$$

- The expected (average) running time is

$$E[\,T(n)\,] = E[\,(\text{time to reduce the array to} \leq 3n/4) + T(3n/4)\,]$$
$$= E[\,(\text{time to reduce the array to} \leq 3n/4)\,] + E[\,T(3n/4)\,]$$

- Since v is "reasonably good" with probability $1/2$ each round, by Discrete Mathematics, the algorithm gets a "reasonably good" v in every two rounds on average.
- Since process one v takes $O(n)$ time, we conclude that

$$E[\,(\text{time to reduce the array to} \leq 3n/4)\,] = O(n)$$

- So $E[\,T(n)\,] = O(n) + E[\,T(3n/4)\,]$.

Alternative Analysis (from Textbook)

- In general, we can reason about the running time as follows:

$$T(n) = (\text{time to reduce the array to} \leq 3n/4) + T(3n/4)$$

- The expected (average) running time is

$$E[\,T(n)\,] = E[\,(\text{time to reduce the array to} \leq 3n/4) + T(3n/4)\,]$$
$$= E[\,(\text{time to reduce the array to} \leq 3n/4)\,] + E[\,T(3n/4)\,]$$

- Since v is "reasonably good" with probability $1/2$ each round, by Discrete Mathematics, the algorithm gets a "reasonably good" v in every two rounds on average.
- Since process one v takes $O(n)$ time, we conclude that

$$E[\,(\text{time to reduce the array to} \leq 3n/4)\,] = O(n)$$

- So $E[\,T(n)\,] = O(n) + E[\,T(3n/4)\,]$.
- Let $E[\,T(n)\,] = A(n)$, we have $A(n) = A(3n/4) + O(n) = \ldots = O(n)$.