

Linear Sorting

Curs: Spring 2019

Upper and lower bounds on time complexity of a problem.

A problem has a **time upper bound** $T_U(n)$ if there is an algorithm A such that **for any input** e of size n :
 $A(e)$ gives the correct answer in $\leq T_U(n)$ steps.

A problem has a **time lower bound** $T_L(n)$ if **there is NO** algorithm which solves the problem if time $< T_L(n)$, **for any input** $e, |e| = n$.

It may be that an algorithm solves the problem faster than $T_L(n)$ for a specific input.

Lower bounds are hard to prove, as we have to consider every possible algorithm.

Upper and lower bounds on time complexity of a problem.

- ▶ Upper bound: $\exists A, \forall e \text{ time } A(e) \leq T_U(|e|)$,
- ▶ Lower bound: $\forall A, \exists e \text{ time } A(e) \geq T_L(|e|)$,

To prove an upper bound: produce an A which works for any e ,
 $|e| = n$.

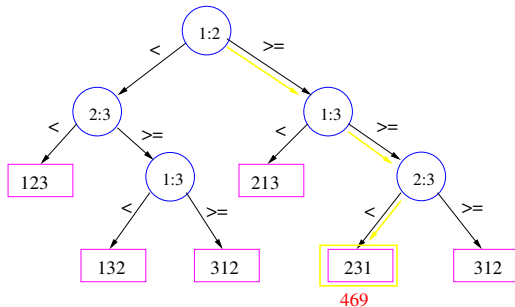
To prove a lower bound, show that **for any possible algorithm**, the time on an input is greater than the lower bound.

Lower bound for **comparison based** sorting algorithm.

Use a **decision tree**: A binary tree where,

- ▶ each internal node represents a comparison $a_i : a_j$, the left subtree represents the case $a_i \leq a_j$ and the right subtree represents the case $a_i > a_j$
- ▶ each leaf represents one of the **$n!$ possible permutations** $(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)})$. Each of the n permutations must appear as one of the leaves of the tree

9	4	6
1	2	3



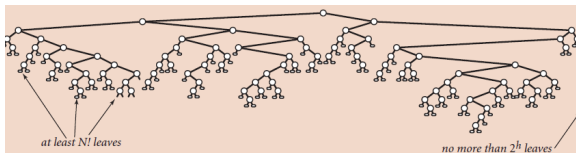
Theorem

Any comparison sort that sorts n elements must perform $\Omega(n \lg n)$ comparisons.

Proof.

Equivalent to prove: Any decision tree that sorts n elements must have height $\Omega(n \lg n)$.

Let h the height of a decision tree with $n!$ leaves,
 $n! \leq 2^h \Rightarrow h \geq \lg(n!) > \lg\left(\frac{n}{e}\right)^n = \Omega(n \lg n)$. □



Linear sorting: Counting sort

CLRS Ch.8

Assume the input $A[1 \dots n]$, is an array of integers in $[0, b]$.

Need: $B[1 \dots n]$ as the output and $C[0 \dots b]$ as scratch.

Counting (A, b)

for $i = 0$ to b **do**

do $C[i] := 0$

end for

for $i = 1$ to n **do**

do $C[A[i]] := C[A[i]] + 1$

end for

for $i = 0$ to b **do**

do $C[i] := C[i] + C[i - 1]$

end for

for $i = n$ downto 1 **do**

do $B[C[A[i]]] := A[i]$

$C[A[i]] := C[A[i]] - 1$

end for

Counting (A, b)

for $i = 0$ to b **do**

do $C[i] := 0$ { $O(b)$ }

end for

for $i = 1$ to n **do**

do $C[A[i]] := C[A[i]] + 1$ { $O(n)$ }

end for

for $i = 0$ to b **do**

do $C[i] := C[i] + C[i - 1]$ { $O(b)$ }

end for

for $i = n$ down to 1 **do**

$B[C[A[i]]] := A[i]$ { $O(n)$ }

$C[A[i]] := C[A[i]] - 1$

end for

Complexity: $T(n) = O(n + b)$ if $b = O(n)$, then $T(n) = O(n)$.

When using counting on short arrays, $b = 10$.

What does it mean radix?

Radix means the base in which we express an integer

Radix 10=Decimal; Radix 2= Binary; Radix 16=Hexadecimal;

Radix 20 (The Maya numerical system)

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Radix Change: Example

- ▶ To convert an enter from binary \rightarrow decimal:
 $1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 12 \Rightarrow 11$
- ▶ To convert an enter from decimal \rightarrow binary: Repeatedly dividing the enter by 2 will give a result plus a remainder:
 $19 \Rightarrow \underbrace{19/2}_{1} \underbrace{9/2}_{1} \underbrace{4/2}_{0} \underbrace{2/2}_{01} = 10011$
- ▶ To transform an integer radix 16 to decimal:
 $(4CF5)_{16} = (4 \times 16^3 + 12 \times 16^2 + 15 \times 16^1 + 5 \times 16^0) = 19701$

radix 10	radix 2	radix 16
7134785012	110101001010001000010110111110100	1a9442df4
4561343780	100001111111000001001010100100100	10fe09524
0051889437	000000011000101111100010100011101	0317c51d

Example

Is it true that if an integer a in radix 512 has d digits, and we change it to be expressed in radix 2, the number of bits that a would have is $\Theta(d^2)$?

NO. When converting radix-256 to binary, we increase d to $\lg 512 \times d = 9d = \Theta(d)$.

Linear sorting: RADIX sort

An important property of counting sort is that it is **stable**, numbers with the same value, appear in the output in the same order as they do in the input.

For instance **Heap sort or Quicksort is not stable.**

Given an array A with n keys, each one with d digits, the Radix (Least Significant Digit),

RADIX LSD (A, d)

for $i = 1$ to d **do**

 Use stable sorting to sort the i -th digit of A .

end for

Example

329		720		720		329
475		475		329		355
657		355		436		436
839	\Rightarrow	436	\Rightarrow	839	\Rightarrow	457
436		657		355		657
720		329		657		720
355		839		475		839

Theorem (Correctness of RADIX)

The previous algorithm sort correctly n keys.

Induction on d .

If $d = 1$ the stable sorting works. Assume it is true for $d - 1$,
to sort the d -th digit,

if $a_d < b_d$ then a will be placed before b ,

if $b_d < a_d$ then b will be placed before a ,

if $b_d = a_d$ then as we are using a stable sorting a and b will remain
in the same order, which by hypothesis was already the correct
one. □

Complexity of RADIX

Given n integers ≥ 0 , each integer with at most d digits, and each digit in the range 0 to 9, if we use counting sorting:

$$T(n, d) = \Theta(d(n + 9)).$$

- ▶ Consider that each integer has a value up to $f(n)$.
- ▶ Then the number of digits is $d = \log f(n)$, so
$$T(n, d) = \Theta((\log f(n))(n + 9)),$$
- ▶ if $f(n) = \omega(1)$ then $T(n) = \omega(n)$. So in this case RADIX is not linear.

Can we do it faster?: Yes, change the radix to express the integers.

RADIX

Given n integers, each integer with a value up to $f(n)$ then the decimal expression had at most $d = \lfloor \log f(n) \rfloor + 1 \sim \log f(n)$ digits.

Consider each integers in radix b , then the number of "digits" is $d = \log_b f(n)$, and apply RADIX to the columns of new digits in base b .

Complexity: Given n positive integers, each integer with a maximal value of $f(n)$ and with at most d digits needed to represent each using radix b . Then the complexity $T(n)$ of using RADIX to sort the n integers is: $T(n) = O((n + b)d) = O(n + b) \log_b f(n)$.

RADIX Change: Example

RADIX 10	RADIX 2	RADIX 16
7134785012	1101010010100010000101110111110100	1a9442df4
4561343780	100001111111000001001010100100100	10fe09524
0051889437	000000011000101111100010100011101	0317c51d

	n	d	b
RADIX 10	3	10	9
RADIX 2	3	33	1
RADIX 16	3	8	15

Implementation for bit integers

Given n integers given as binary strings with length d , we want to choose an integer $1 < r < d$ such minimizes $(d/r)(n + 2^r)$.

Then use RADIX on each of the new $\hat{d} = \lceil d/r \rceil$ digits.

For ex., if we have words of $b = 32$ bits, which we split in $d = 4$ $r = 8$ -bit digits:

1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0

Each new digit is an integer with $b = 2^r - 1$.

So we can use counting sort with $k = 2^r - 1$.

Each pass of counting sort takes $\Theta(n + k) = \Theta(n + 2^r)$, as there are d passes $\Rightarrow T(n) = \Theta(d(n + 2^r)) = \Theta((b/r)(n + 2^r))$.

Comparing radix and counting:

For n integers, each integer with at most d digits, where each digit is in the range $[0, 9]$:

- ▶ Counting sort is $\Theta(9dn)$,
- ▶ Radix with the choice of $r = \log_9 n$ -digits can sort n d -digit numbers in $\frac{d}{\log_9 n} \Theta(9n)$.

Consider 2000 integers of 32 bits each:

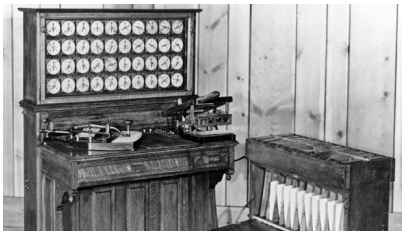
- ▶ Quicksort needs to do $\lg 2000 = 11$ passes over the data,
- ▶ Radix sort with digits of 11-bits, takes 3 passes (at each one counting sort makes 2 passes).

Empirically, when dealing with natural numbers, radix is better than other sorting methods for values of $n > 2000$.

A bit of history.

Radix and all counting sort are due to Herman Hollerith.

In 1890 he invented the card sorter that allowed to shorten the US census to 5 weeks, using punching cards.



1	A	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	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----