

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on May 28, 2023)

1. Primitive sorting networks at random. This program is a quick-and-dirty implementation of the random process studied in exercise 5.3.4–40: Start with the permutation $n \dots 21$ and randomly interchange adjacent elements that are out of order, until reaching $12 \dots n$. I want to know if the upper bound of $4n^2$ steps, proved in that exercise, is optimum.

This Monte Carlo program computes a number c such that $c(n-1)$ random adjacent comparators would have sufficed to complete the sorting. This number is the sum of $1/t_k$ during the $\binom{n}{2}$ steps of sorting, where t is the number of adjacent out-of-order pairs before the k th step. If c is consistently less than $4n$, the exercise's upper bound is too high.

In fact, ten experiments with $n = 10000$ all gave $19904 < c < 20017$; hence it is extremely likely that the true asymptotic behavior is $\sim 2n^2$.

```
#include <stdio.h>
#include <math.h>
#include "gb_flip.h"
int *perm;
int *list;
int seed; /* random number seed */
int n; /* this many elements */
main(argc, argv)
    int argc;
    char *argv[];
{
    register int i, j, k, t, x, y;
    register double s;
    <Scan the command line 2>;
    <Initialize everything 3>;
    while (t) <Move 4>;
    <Print the results 5>;
}

2. <Scan the command line 2> ≡
if (argc ≠ 3 ∨ sscanf(argv[1], "%d", &n) ≠ 1 ∨ sscanf(argv[2], "%d", &seed) ≠ 1) {
    fprintf(stderr, "Usage: %s %d %d\n", argv[0]);
    exit(-1);
}
```

This code is used in section 1.

3. We maintain the following invariants: the indices i where $perm[i] > perm[i+1]$ are $list[j]$ for $0 \leq j < t$.

```
<Initialize everything 3> ≡
gb_init_rand(seed);
perm = (int *) malloc(4 * (n + 2));
list = (int *) malloc(4 * (n - 1));
for (k = 1; k ≤ n; k++) perm[k] = n + 1 - k;
perm[0] = 0; perm[n + 1] = n + 1;
for (k = 1; k < n; k++) list[k - 1] = k;
t = n - 1;
s = 0.0;
```

This code is used in section 1.

4. $\langle \text{Move } 4 \rangle \equiv$

```

{
  s += 1.0/((double) t);
  j = gb_unif_rand(t);
  i = list[j];
  t--;
  list[j] = list[t];
  x = perm[i]; y = perm[i + 1];
  perm[i] = y; perm[i + 1] = x;
  if (perm[i - 1] > y ∧ perm[i - 1] < x) list[t++] = i - 1;
  if (perm[i + 2] < x ∧ perm[i + 2] > y) list[t++] = i + 1;
}

```

This code is used in section 1.

5. Is this program simple, or what?

$\langle \text{Print the results } 5 \rangle \equiv$

```

printf("%g□=□%g\n", s, s/((double) n);

```

This code is used in section 1.

6. Index.

argc: [1](#), [2](#).
argv: [1](#), [2](#).
exit: [2](#).
fprintf: [2](#).
gb_init_rand: [3](#).
gb_unif_rand: [4](#).
i: [1](#).
j: [1](#).
k: [1](#).
list: [1](#), [3](#), [4](#).
main: [1](#).
malloc: [3](#).
n: [1](#).
perm: [1](#), [3](#), [4](#).
printf: [5](#).
s: [1](#).
seed: [1](#), [2](#), [3](#).
sscanf: [2](#).
stderr: [2](#).
t: [1](#).
x: [1](#).
y: [1](#).

⟨ Initialize everything 3 ⟩ Used in section 1.
⟨ Move 4 ⟩ Used in section 1.
⟨ Print the results 5 ⟩ Used in section 1.
⟨ Scan the command line 2 ⟩ Used in section 1.

RAN-PRIM

	Section	Page
Primitive sorting networks at random	1	1
Index	6	3