

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

1. Introduction. This is a quick-and-dirty implementation of the Garsia-Wachs algorithm, written as I was preparing the 2nd edition of Volume 3, then patched after Wolfgang Panny discovered a serious bug. (The bug was corrected in the 17th printing of the 2nd edition, October 2004.)

The input weights are given on the command line.

The leaf nodes are $0, 1, \dots, n$; the internal nodes are $n + 1, n + 2, \dots, 2n$.

```
#define size 64      /* this number should exceed twice the number of input weights */
#include <stdio.h>
int w[size];        /* node weights */
int l[size], r[size]; /* left and right children */
int d[size];        /* depth */
int q[size];        /* working region */
int v[size];        /* number of node in working region */
int t;              /* current size of working region */
int m;              /* current node */
<Subroutines 4>
main(argc, argv)
    int argc;
    char *argv[];
{
    register int i, j, k, n;
    <Scan the command line 2>;
    <Do phase 1 3>;
    <Do phase 2 5>;
    <Do phase 3 7>;
}
```

2. <Scan the command line 2> \equiv

```
n = argc - 2;
if (n < 0) {
    fprintf(stderr, "Usage: %s s wt0 . . . wt n\n", argv[0]);
    exit(0);
}
if (n + n > size) {
    fprintf(stderr, "Recompile me with a larger tree size!\n");
    exit(0);
}
for (j = 0; j <= n; j++) {
    if (sscanf(argv[j + 1], "%d", &m) != 1) {
        fprintf(stderr, "Couldn't read wt%d!\n", j);
        exit(0);
    }
    w[j] = m;
    l[j] = r[j] = -1;
}
```

This code is used in section 1.

3. $\langle \text{Do phase 1 } 3 \rangle \equiv$

```

printf("Phase_I:\n");
m = n;
t = 1;
q[0] = 1000000000; /* infinity */
q[1] = w[0];
v[1] = 0;
for (k = 1; k ≤ n; k++) {
    while (q[t - 1] ≤ w[k]) combine(t);
    t++;
    q[t] = w[k];
    v[t] = k;
    for (j = 1; j ≤ t; j++) printf("%d_", q[j]);
    printf("\n");
}
while (t > 1) combine(t);

```

This code is used in section 1.

4. The *combine* subroutine combines weights $q[k - 1]$ and $q[k]$ of the working list, and continues to combine earlier weights if necessary to maintain the condition $q[j - 1] > q[j + 1]$.

(The bug in my previous version was, in essence, to use ‘**if**’ instead of ‘**while**’ in the final statement of this routine.)

$\langle \text{Subroutines } 4 \rangle \equiv$

```

combine(register int k)
{
    register int j, d, x;
    m++;
    l[m] = v[k - 1];
    r[m] = v[k];
    w[m] = x = q[k - 1] + q[k];
    printf("_node_%d(%d)=%d(%d)+%d(%d)\n", m, x, l[m], w[l[m]], r[m], w[r[m]]);
    t--;
    for (j = k; j ≤ t; j++) q[j] = q[j + 1], v[j] = v[j + 1];
    for (j = k - 2; q[j] < x; j--) q[j + 1] = q[j], v[j + 1] = v[j];
    q[j + 1] = x;
    v[j + 1] = m;
    for (d = 1; d ≤ t; d++) printf("%d_", q[d]);
    printf("\n");
    while (j > 0 ∧ q[j - 1] ≤ x) {
        d = t - j;
        combine(j);
        j = t - d;
    }
}

```

See also sections 6 and 8.

This code is used in section 1.

5. $\langle \text{Do phase 2 } 5 \rangle \equiv$

```

printf("Phase_II:\n");
mark(v[1], 0);

```

This code is used in section 1.

6. The *mark* subroutine assigns level numbers to a subtree.

⟨Subroutines 4⟩ +≡

```
mark(k, p)
    int k;    /* node */
    int p;    /* starting depth */
{
    printf("node%d(%d) has depth%d\n", k, w[k], p);
    d[k] = p;
    if (l[k] ≥ 0) mark(l[k], p + 1);
    if (r[k] ≥ 0) mark(r[k], p + 1);
}
```

7. ⟨Do phase 3 7⟩ ≡

```
printf("Phase_III:\n");
t = 0;
m = 2 * n;
build(1);
```

This code is used in section 1.

8. The *build* subroutine rebuilds a tree from the depth table, by doing a depth-first search according a slick idea by Bob Tarjan. It creates a tree rooted at node *m* having leftmost leaf *t*.

⟨Subroutines 4⟩ +≡

```
build(b)
    int b;    /* depth of node m, plus 1 */
{
    register int j = m;
    if (d[t] ≡ b) l[j] = t++;
    else {
        m--;
        l[j] = m;
        build(b + 1);
    }
    if (d[t] ≡ b) r[j] = t++;
    else {
        m--;
        r[j] = m;
        build(b + 1);
    }
    printf("node%d=%d+%d\n", j, l[j], r[j]);
}
```

9. Index.*argc*: [1](#), [2](#).*argv*: [1](#), [2](#).*b*: [8](#).*build*: [7](#), [8](#).*combine*: [3](#), [4](#).*d*: [1](#), [4](#).*exit*: [2](#).*fprintf*: [2](#).*i*: [1](#).*j*: [1](#), [4](#), [8](#).*k*: [1](#), [4](#), [6](#).*l*: [1](#).*m*: [1](#).*main*: [1](#).*mark*: [5](#), [6](#).*n*: [1](#).*p*: [6](#).*printf*: [3](#), [4](#), [5](#), [6](#), [7](#), [8](#).*q*: [1](#).*r*: [1](#).*size*: [1](#), [2](#).*sscanf*: [2](#).*stderr*: [2](#).*t*: [1](#).*v*: [1](#).*w*: [1](#).*x*: [4](#).

- ⟨ Do phase 1 [3](#) ⟩ Used in section [1](#).
- ⟨ Do phase 2 [5](#) ⟩ Used in section [1](#).
- ⟨ Do phase 3 [7](#) ⟩ Used in section [1](#).
- ⟨ Scan the command line [2](#) ⟩ Used in section [1](#).
- ⟨ Subroutines [4](#), [6](#), [8](#) ⟩ Used in section [1](#).

GARSIA-WACHS

	Section	Page
Introduction	1	1
Index	9	4