DESIGN.pdf

Githika Annapureddy

February 2023

1 Shell Sort

When this function swaps, compares, or moves elements in the array, the functions swap(), cmp(), or move() from stats.c are used.
shell_sort takes in Stats, *arr, and length as arguments.
This function uses the gaps array created in gaps.h.
gap = gaps[n]
while gap ¿ 0
for int i in range(0, gaps)
swap = i
temp = arr[i]
while(counter ¿= gap and arr[swap - gap] ¿ temp)
arr[swap] = arr[swap - gap]
move(stats, arr[swap])
swap -= gap
arr[swap] = temp
move(stats, arr[swap])
iterate to next gap in gaps

2 Heap Sort

Heap Sort has 4 functions. Each function is called by another. When a function swaps, compares, or moves elements in the array, the functions swap(), cmp(), or move() from stats.c are used.

max_child takes arguments *stats, A[], first, last
left = 2 * first;
right = left + 1;
compare;
compare = cmp(stats, A[right - 1], A[left - 1]);
if (right ¡= last (compare == 1)) //use cmp() to do A[right - 1] ¿ A[left - 1]
return right
return left

fix_heap takes arguments *stats, Array[], first, last
found = 0;
compare;
mother = first;
great = max_child(stats, Array, mother, last);

while (mother ¡= (last / 2) !(found))
compare = cmp(stats, Array[mother - 1], Array[great - 1]);
if (compare == -1)
swap(stats, Array[mother -1], Array[great -1]);
mother = great;
great = max_child(stats, Array, mother, last);

else
found = 1

build_heap takes arguments *stats, Array[], first, last

for (father = (last/2); father ¿ (first - 1); father–)
fix_heap(stats, Array, father, last)

heap_sort takes arguments stats, *array, length

first = 1
last = length
build_heap(stats, A, first, last)
for (int leaf = last; leaf ¿ first; leaf–)

swap(stats, A[first -1], A[leaf-1])
fix_heap(stats, A, first, leaf - 1)

3 Quick Sort

Quick Sort has 2 functions. Each function is called by another. When a function swaps, compares, or moves elements in the array, the functions swap(), cmp(), or move() from stats.c are used.
partition takes arguments *stats, A[], lo, hi)
i = lo - 1;
compare;
for (j = lo; j ¡ hi; j++)
compare = cmp(stats, A[j - 1], A[hi - 1]);

if (compare == -1) //uses cmp() to do A[j - 1] ¡ A[hi - 1]
i++;
//swap A[j - 1] and A[i - 1]
swap(stats, A[j - 1], A[i - 1]);
//swap A[i] and A[hi - 1]
swap(stats, A[i], A[hi - 1]);
return (i + 1);

quick$_s ortertakesarguments * stats, A[], lo, hi)$
p;
$if(lo < hi)$
$p = partition(stats, A, lo, hi)$;
$quick_s orter(stats, A, lo, p - 1)$;
$quick_s orter(stats, A, p + 1, hi)$;

$quick_s orttakesarguments * stats, *A, n)$
$quick_s orter(stats, A, 1, n)$;

4 Batcher Sort

Batcher Sort has 2 functions. Each function is called by another. When a function swaps, compares, or moves elements in the array, the functions swap(), cmp(), or move() from stats.c are used.
comparator takes arguments *stats, A[], x, y)
compare = cmp(stats, A[x], A[y]);
if (compare == 1) //use cmp() for A[x] ¿ A[y]
swap(stats, A[x], A[y]); //use swap()

batcher_sort takes arguments *stats, *A, n)
if (n == 0)
return; //terminates function
while (p ¿ 0)
q = 1 ¡¡ (t - 1);
r = 0;
d = p;
while (d ¿ 0)
for (int i = 0; i + d ¡ n; i++)
if ((i p) == r)
comparator(stats, A, i, i + d);
d = q - p;
q = q ¿¿ 1;
r = p;
p = p ¿¿ 1;

5 Sorting.c (Test)

Sorting.c uses the main function and 4 helper functions to print each sort appropriately.
//helper functions to print the appropriate responses
Each helper function uses the Stats Struct.
shellsort takes arguments A[], length, elements
Stats shell$_s tat$;
$shell_s tat.moves = 0$;
$shell_s tat.compares = 0$;
$shell_s ort(shell_s tat, A, length)$;
$printf("HeapSort(length)elements, (heap_s tat.moves)moves, heap_s tat.compares)compares")$
$printeachelementandprintanewlineevery5elements$

$quicksorttakesargumentsA[], length, elements$
$Statsquick_s tat$;
$quick_s tat.moves = 0$;
$quick_s tat.compares = 0$;
$quick_s ort(quick_s tat, A, length)$;
$printf("HeapSort(length)elements, (heap_s tat.moves)moves, (heap_s tat.compares)compares")$
$printeachelementandprintanewlineevery5elements$

$batchersorttakesargumentsA[], length, elements$
$Statsbatcher_s tat$;
$batcher_s tat.moves = 0$;
$batcher_s tat.compares = 0$;
$batcher_s ort(batcher_s tat, A, length)$;
$printf("HeapSort(length)elements, (heap_s tat.moves)moves, (heap_s tat.compares)compares")$
$printeachelementandprintanewlineevery5elements$

$heapsorttakesargumentsA[], length, elements$
$Statsheap_s tat$;
$heap_s tat.moves = 0$;
$heap_s tat.compares = 0$;
$heap_s ort(heap_s tat, A, length)$;
$printf("HeapSort(length)elements, (heap_s tat.moves)moves, (heap_s tat.compares)compares")$
$printeachelementandprintanewlineevery5elements$

$mainfunctiontakesargumentsargc, **argv$
$opt = 0$;
$seed = 13371453$;
$size = 100$;
$elements = 100$;

$while((opt = getopt(argc, argv, OPTIONS))! = -1)$

$switch(opt)$
$case'a' : a = 1;$
$break;$

case 'h': h = 1;
break;

case 'b': b = 1;
break;

case 's': s = 1;
break;

case 'q': q = 1;
break;

case 'r': seed = (uint32$_t$)$strtoul(optarg, NULL, 10);$;
$break;$

case 'n': size = (uint32$_t$)$strtoul(optarg, NULL, 10); elements = size;$
$break;$

case 'p': elements = (uint32$_t$)$strtoul(optarg, NULL, 10);$
$break;$

case 'H': printf("usage message");
break;

srandom(seed);
uint32$_t A[size];$
$for(uint32_t i = 0; i < size; i++)$
$A[i] = random();$

if (a == 1)

shellsort(A, size, elements);
quicksort(A, size, elements);
heapsort(A, size, elements);
batchersort(A, size, elements);

if(b ==1)
batchersort(A, size, elements);

if(s == 1)
shellsort(A, size, elements);

```
if(q == 1)
quicksort(A, size, elements);

    if(h == 1)
heapsort(A, size, elements);
return 0;
```