```
1 /* TBB */
2 class Sort: public task {
3 public:
4  int* A; int* tmp; int size; int cutoff;
5  Sort(int* _A, int* _tmp, int _size, int _cutoff):
6   A(_A), tmp(_tmp), size(_size), cutoff(_cutoff){}
7  task* execute() {
8   if (cutoff==1) {
9     SeqSort(A, tmp, size);
10  } else {
11    Sort& a = *new(allocate_child()) Sort(A, tmpA,
          half, cutoff/2);
12    Sort& b = *new(allocate_child()) Sort(B, tmpB,
          size-half, cutoff/2);
13    set_ref_count(3);
14    spawn(a);
15    spawn_and_wait_for_all(b);
16    Merge& c = *new(allocate_child()) Merge(A, B,
      tmp, size);
17    set_ref_count(2);
18    spawn_and_wait_for_all(c);
19  }
20  return NULL;}
21 };
22
23 //The other three approaches use the function Sort
24 void Sort(int* A, int* tmp, int size, int cutoff){
25  int half = size/2;
26  int* tmpA = tmp;
27  int* B = A + half;
28  int* tmpB = tmpA + half;
29
30 /* OpenMP */
31  if (cutoff==1) {
32    SeqSort(A, tmp, size);
33  } else {
34   #pragma omp task
35    Sort(A, tmpA, half, cutoff/2);
36   #pragma omp task
37    Sort(B, tmpB, size-half, cutoff/2);
38   #pragma omp taskwait
39   #pragma omp task
40    Merge(A, B, tmp, size);
41   #pragma omp taskwait
42  }
43
44 /* Cilk Plus */
45  if (cutoff==1) {
46    SeqSort(A, tmp, size);
47  } else {
48    _Cilk_spawn Sort(A, tmpA, half, cutoff/2);
49    _Cilk_spawn Sort(B, tmpB, size-half,cutoff/2);
50    _Cilk_sync;
51    _Cilk_spawn Merge(A, B, tmp, size);
52    _Cilk_sync;
53  }
54 }
```