Description:

C code implementation using Divide and Conquer algorithm to find the convex hull of a given set of 2D points and CPU time.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Structure to represent a point

typedef struct

{

int first;

int second;

} Pair;


// Global variable to store the center of the polygon

Pair mid;


// Function to determine the quadrant of a point

int quad(Pair p)

{

if (p.first >= 0 && p.second >= 0)

return 1;

if (p.first <= 0 && p.second >= 0)

return 2;

if (p.first <= 0 && p.second <= 0)
```

```
return 3;

return 4;

}


// Function to check the orientation of three points

int orientation(Pair a, Pair b, Pair c)

{

int res = (b.second - a.second) * (c.first - b.first) - (c.second - b.second) * (b.first - a.first);


if (res == 0)

return 0;

if (res > 0)

return 1;

return -1;

}


// Compare function for sorting

int compare(const void *p1, const void *q1)

{

Pair *p = (Pair *)p1;

Pair *q = (Pair *)q1;


Pair p_diff = {p->first - mid.first, p->second - mid.second};

Pair q_diff = {q->first - mid.first, q->second - mid.second};


int one = quad(p_diff);
int two = quad(q_diff);

if (one != two)
return (one < two) ? -1 : 1;

return (p_diff.second * q_diff.first < q_diff.second * p_diff.first) ? -1 : 1;
}

// Function to merge two convex hulls
Pair *merger(Pair *a, int n1, Pair *b, int n2, int *ret_size)
{
int ia = 0, ib = 0,i;
for (i = 1; i < n1; i++)
if (a[i].first > a[ia].first)
ia = i;

for (i = 1; i < n2; i++)
if (b[i].first < b[ib].first)
ib = i;

int inda = ia, indb = ib;
int done = 0;
while (!done)
{
done = 1;
while (orientation(b[indb], a[inda], a[(inda + 1) % n1]) >= 0)
inda = (inda + 1) % n1;

while (orientation(a[inda], b[indb], b[(n2 + indb - 1) % n2]) <= 0)
{

indb = (n2 + indb - 1) % n2;
done = 0;
}
}

int uppera = inda, upperb = indb;
inda = ia;
indb = ib;
done = 0;
while (!done)
{
done = 1;
while (orientation(a[inda], b[indb], b[(indb + 1) % n2]) >= 0)
indb = (indb + 1) % n2;

while (orientation(b[indb], a[inda], a[(n1 + inda - 1) % n1]) <= 0)
{
inda = (n1 + inda - 1) % n1;
done = 0;
}
}
```

```
int lowera = inda, lowerb = indb;

Pair *ret = (Pair *)malloc((n1 + n2) * sizeof(Pair));

int ind = uppera;

int k = 0;


ret[k++] = a[uppera];

while (ind != lowera)

{

ind = (ind + 1) % n1;

ret[k++] = a[ind];

}


ind = lowerb;

ret[k++] = b[lowerb];

while (ind != upperb)

{

ind = (ind + 1) % n2;

ret[k++] = b[ind];

}


*ret_size = k;

return ret;

}


// Brute force algorithm to find the convex hull for a small set of points
```

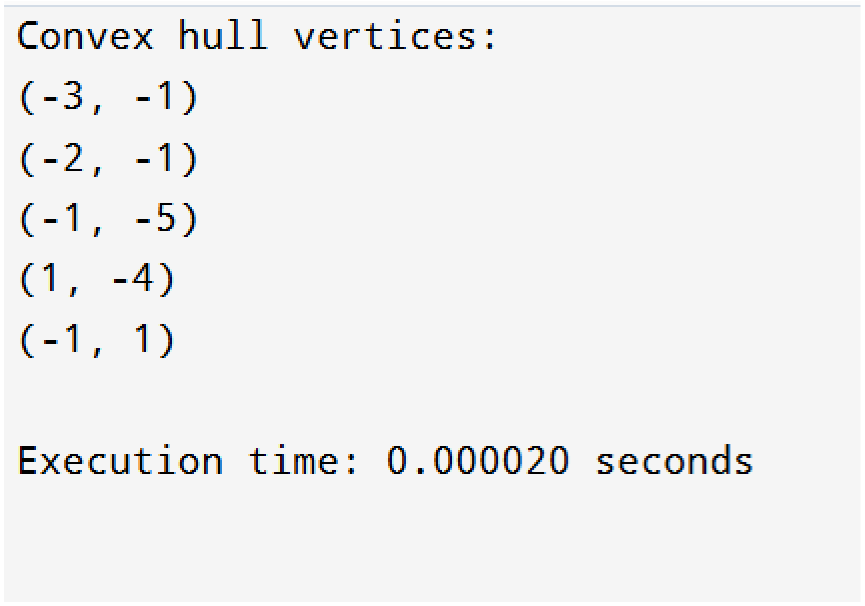
```
Pair *bruteHull(Pair *a, int n, int *ret_size)

{

int max_combinations = n * (n - 1) / 2;

Pair *ret = (Pair *)malloc(n * sizeof(Pair));

int k = 0;

int j;

for (i = 0; i < n; i++)

{

for (j = i + 1; j < n; j++)

{

int x1 = a[i].first, x2 = a[j].first;

int y1 = a[i].second, y2 = a[j].second;


int a1 = y1 - y2;

int b1 = x2 - x1;

int c1 = x1 * y2 - y1 * x2;

int pos = 0, neg = 0;


for (int m = 0; m < n; m++)

{

if (a1 * a[m].first + b1 * a[m].second + c1 <= 0)

neg++;

if (a1 * a[m].first + b1 * a[m].second + c1 >= 0)

pos++;

}
if (pos == n || neg == n)
{
int already_added = 0;
for (int l = 0; l < k; l++)
{

if (ret[l].first == a[i].first && ret[l].second == a[i].second)
already_added = 1;
}
if (!already_added)
ret[k++] = a[i];

already_added = 0;
for (int l = 0; l < k; l++)
{
if (ret[l].first == a[j].first && ret[l].second == a[j].second)
already_added = 1;
}
if (!already_added)
ret[k++] = a[j];
}
}
}

*ret_size = k;
ret = (Pair *)realloc(ret, k * sizeof(Pair));
```

```
mid.first = 0;

mid.second = 0;

for (int i = 0; i < k; i++)

{

mid.first += ret[i].first;


mid.second += ret[i].second;

ret[i].first *= k;

ret[i].second *= k;

}


qsort(ret, k, sizeof(Pair), compare);


for (int i = 0; i < k; i++)

{

ret[i].first /= k;

ret[i].second /= k;

}


return ret;

}


// Function to divide the set of points and recursively find the convex hull

Pair *divide(Pair *a, int n, int *ret_size)

{
```

```
if (n <= 5)
return bruteHull(a, n, ret_size);

int mid = n / 2;
Pair *left_hull;
Pair *right_hull;
int left_size, right_size;

left_hull = divide(a, mid, &left_size);
right_hull = divide(a + mid, n - mid, &right_size);

return merger(left_hull, left_size, right_hull, right_size, ret_size);
}

// Driver code
int main()
{
Pair a[] = {{0, 0}, {1, -4}, {-1, -5}, {-5, -3}, {-3, -1},
{-1, -3}, {-2, -2}, {-1, -1}, {-2, -1}, {-1, 1}};
clock_t start,end;
start=clock();
int n = sizeof(a) / sizeof(a[0]);

qsort(a, n, sizeof(Pair), compare);
int ret_size;
Pair *ans = divide(a, n, &ret_size);
```

```
printf("Convex hull:\n");

for (int i = 0; i < ret_size; i++)

{

printf("%d %d\n", ans[i].first, ans[i].second);

}

end=clock();

double cpu_time_used;

cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;

printf("%2f is the execution time");

free(ans);

return 0;

}
```

Output:



```
Convex hull vertices:
(-3, -1)
(-2, -1)
(-1, -5)
(1, -4)
(-1, 1)

Execution time: 0.000020 seconds
```